

RPC Broker 1.1

User Guide (REDACTED)



September 2021

Department of Veterans Affairs (VA)
Office of Information and Technology (OIT)
Enterprise Program Management Office (EPMO)

Revision History

Document Revisions

Date	Revision	Description	Authors
09/15/2021	10.0	<p>Tech Edits based on the Broker Development Kit (BDK) release with RPC Broker Patch XWB*1.1*73 (Client-Side only; no Vista M Server updates):</p> <ul style="list-style-type: none"> • Supports Delphi XE8, 10.0, 10.1, 10.2, 10.3, and Delphi/RAD Studio v10.4: Sections 1.1.1 and 7.1. • Corrects the following issues: <ul style="list-style-type: none"> ○ Ensures the data placed into the Brokerx.User.Division field is correctly formatted. ○ Redesigned the method of certificate processing; it automatically selects the user's Authentication certificate, eliminating the need for the user to select from a list of certificates. • Added the ShowCertDialog property. • Deleted references to online help and .chm file, since the online help is <i>not</i> being released with RPC Broker Patch XWB*1.1*73. <p>RPC Broker 1.1; XWB*1.1*73 BDK</p>	RPC Broker Development Team
12/17/2020	9.0	<p>Tech Edits based on the Broker Development Kit (BDK) release with RPC Broker Patch XWB*1.1*72 (Client-Side only; no Vista M Server updates):</p> <ul style="list-style-type: none"> • Supports Delphi XE8, 10.0, 10.1, 10.2, 10.3, and Delphi/RAD Studio v10.4: Sections 1.1.1 and 7.1. • Corrects the following issues: <ul style="list-style-type: none"> ○ Ensures the DIVISION field is properly set. ○ Addresses Hints and Warnings along with many of the 	RPC Broker Development Team

Date	Revision	Description	Authors
		memory leaks. RPC Broker 1.1; XWB*1.1*72 BDK	
05/06/2020	8.0	<p>Tech Edits based on the Broker Development Kit (BDK) release with RPC Broker Patch XWB*1.1*71.</p> <ul style="list-style-type: none"> Updated Section 1.1.1 functionality added with XWB*1.1*71 Changed all references throughout to “Patch XWB*1.1*71” as the latest BDK release. Updated references to show RPC Broker Patch XWB*1.1*71 supports Delphi 10.3, 10.2, 10.1, 10.0, and XE8 throughout. This was a bug fix-only patch, so no new options, routines, files, fields, security keys, APIs, or RPCs. Reformatted all references to file and field name numbers throughout. Updated all styles and formatting to match current documentation standards and style guidelines. <p>RPC Broker 1.1; XWB*1.1*71 BDK</p>	RPC Broker Development Team
05/18/2017	7.2	<p>Updated the CALLBACKTYPE entry in “Table 10: Fields in the REMOTE APPLICATION (#8994.5) File” to include the “S—Station-number callback” value.</p> <p>RPC Broker 1.1; XWB*1.1*65 BDK</p>	RPC Broker Development Team
05/17/2017	7.1	<p>Tech Edits:</p> <ul style="list-style-type: none"> Updated/Added “Caution” note for the Reference PType input parameter in Table 6, Step 1 in Section 3.6, and Section 4.3. Reformatted all references to file and field name numbers throughout. <p>RPC Broker 1.1; XWB*1.1*65 BDK</p>	RPC Broker Development Team
01/24/2017	7.0	<p>Tech Edits based on release of RPC Broker Patch XWB*1.1*65:</p>	RPC Broker Development Team

Date	Revision	Description	Authors
		<ul style="list-style-type: none"> Reformatted document to follow current documentation standards and style formatting requirements. <p>Inserted Section 5, "Broker Security Enhancement (BSE);" content taken from Chapters 1-2 in the Broker Security Enhancement (BSE) Patch XWB*1.1*45 Supplement.</p> <ul style="list-style-type: none"> Added content and references to the TXWBSSOi component in Sections 1.1 and 2.4. Updated Section 1.1.1 for 2-factor authentication feature and current level of Delphi version support. Updated Section 2.1.4. Added Caution note to the Reference PType in Table 6. Updated Figure 6. Updated registry information in Section 4.1.1. Added Figure 8. Corrected Section 4.1.2. Updated debug instructions in Section 6.1. Updated instructions in Section 6.2.1. Updated Section 7.1 and 7.1.1 for currently supported Delphi versions. Updated Section 7.1.2 and 7.1.3 for .bpl file references. Changed references from "Borland Delphi" to "Embarcadero Delphi" throughout. Added new glossary terms: SAML and XML. <p>RPC Broker 1.1; XWB*1.1*65 BDK</p>	
04/27/2016	6.0	<p>Tech Edits:</p> <ul style="list-style-type: none"> Reformatted document to 	RPC Broker Development Team

Date	Revision	Description	Authors
		<p>follow current documentation standards and style formatting requirements.</p> <ul style="list-style-type: none"> Updated the "Orientation" section. Updated Section 1.1.1. Updated Table 3 for TRPCBroker component key properties. Updated Section 2.1.4. Updated Figure 1. Deleted Sections 2.3, "TSharedBroker Component" and 2.4, "TSharedRPCBroker Component." Updated Section 3.2. Added Section 3.2.1 and titled and modified Section 3.2.2. Updated Table 7. Updated Section 3.7.2. Updated Figure 6. Updated Section 4.1. Updated Figure 7. Updated Section 4.1.2. Update Figure 9. Updated Sections 6.2.1 and 6.2.2. Updated Section 7. Updated Sections 7.1.1, 7.1.2, and 7.1.3. Deleted, Sections 6.1.4, "SharedRPCBroker_RXE5.bpl File" and 6.1.5, "SharedRPCBroker_DXE5.bpl File." Deleted Sections 6.2, "Delphi XE4 Packages," 6.3, "Delphi XE3 Packages," and 6.4, "Delphi XE2 Packages." Updated Section 8.1. Deleted references to TSharedRPCBroker and TSharedBroker components 	

Date	Revision	Description	Authors
		<p>throughout, since they were removed from the software.</p> <ul style="list-style-type: none"> Updated help file references from "BROKER.HLP" to "Broker_1_1.chm" throughout. Updated references to show RPC Broker Patch XWB*1.1*60 supports Delphi XE7, XE6, XE5, and XE4 throughout. <p>RPC Broker 1.1; XWB*1.1*60 BDK</p>	
12/04/2013	5.1	<p>Tech Edit:</p> <ul style="list-style-type: none"> Updated document for RPC Broker Patch XWB*1.1*50 based on feedback from the developer. Removed references related to Virgin Installations throughout. Updated file name references throughout. Removed distribution files that are obsolete or no longer distributed throughout. Updated RPC Broker support on the following software: <ul style="list-style-type: none"> Microsoft® XP and 7.0 (operating system) throughout. Microsoft® Office Products 2010 throughout. Changed references from "Borland" to "Embarcadero" and updated support for Delphi Versions XE5, XE4, XE3, and XE2 throughout. Updated all images for prior Microsoft® Windows operating systems to Windows 7 dialogues. Deleted Section 6, "RPC Broker Developer Utilities," since those utilities no longer exist in this latest version of 	RPC Broker Development Team

Date	Revision	Description	Authors
		<p>the Broker.</p> <ul style="list-style-type: none"> Updated the “RPC Broker and Delphi” section for Delphi XE5, XE4, XE3, and XE2. Removed sample DLL from Section 8. Redacted document for the following information: <ul style="list-style-type: none"> Names (replaced with role and initials). Production IP addresses and ports. Intranet websites. <p>RPC Broker 1.1; XWB*1.1*50 BDK</p>	
07/25/2013	5.0	<p>Tech Edit:</p> <ul style="list-style-type: none"> Baselined document. Updated all styles and formatting to follow current internal team style template. Updated all organizational references. <p>RPC Broker 1.1; XWB*1.1*50 BDK</p>	RPC Broker Development Team
08/26/2008	4.2	<p>Updates for RPC Broker Patch XWB*1.1*50:</p> <ul style="list-style-type: none"> Added new properties. Support for Delphi 5, 6, 7, 2005, 2006, and 2007. Changed references from Patch 47 to Patch 50 where appropriate. <p>RPC Broker 1.1; XWB*1.1*50 BDK</p>	RPC Broker Development Team
07/03/2008	4.1	<p>Updates for RPC Broker Patch XWB*1.1*47:</p> <ul style="list-style-type: none"> No content changes required; no new public classes, methods, or properties added to those available in XWB*1.1*40. Bug fixes to the ValidAppHandle function and fixed memory leaks. Support added for Delphi 	RPC Broker Development Team

Date	Revision	Description	Authors
		<p>2005, 2006, and 2007.</p> <ul style="list-style-type: none"> • Reformatted document. • Changed references from Patch 40 to Patch 47 where appropriate. <p>RPC Broker 1.1; XWB*1.1*47 BDK</p>	
02/24/2005	4.0	<p>Revised Version for RPC Broker Patches XWB*1.1*35 and 40.</p> <p>Also, reviewed document and edited for the “Data Scrubbing” and the “PDF 508 Compliance” projects.</p> <p>Data Scrubbing—Changed all patient/user TEST data to conform to standards and conventions as indicated below:</p> <ul style="list-style-type: none"> • The first three digits (prefix) of any Social Security Numbers (SSN) start with “000” or “666.” • Patient or user names are formatted as follows: XWBPATIENT,[N] or XWBUSER,[N] respectively, where the N is a number written out and incremented with each new entry (e.g., XWBPATIENT, ONE, XWBPATIENT, TWO, etc.). • Other personal demographic-related data (e.g., addresses, phones, IP addresses, etc.) were also changed to be generic. <p>PDF 508 Compliance—The final PDF document was recreated and now supports the minimum requirements to be 508 compliant (i.e., accessibility tags, language selection, alternate text for all images/icons, fully functional Web links, successfully passed Adobe Acrobat Quick Check).</p> <p>RPC Broker 1.1; XWB*1.1*35 & 40 BDK</p>	RPC Broker Development Team
05/08/2002	3.0	<p>Revised Version for RPC Broker Patch XWB*1.1*26.</p> <p>RPC Broker 1.1; XWB*1.1*26 BDK</p>	RPC Broker Development Team

Date	Revision	Description	Authors
05/01/2002	2.0	Revised Version for RPC Broker Patch XWB*1.1*13. RPC Broker 1.1; XWB*1.1*13 BDK	RPC Broker Development Team
09/--/1997	1.0	Initial RPC Broker Version 1.1 software release. RPC Broker 1.1	RPC Broker Development Team

Patch Revisions

For the current patch history related to this software, see the Patch Module on FORUM.

Table of Contents

Revision History	ii
List of Figures.....	xii
List of Tables	xii
Orientation	xiv
1 Introduction	1
1.1 About this Version of the BDK	1
1.1.1 Features	2
1.1.2 Backward Compatibility Issues	3
2 RPC Broker Components for Delphi	4
2.1 TRPCBroker Component	4
2.1.1 TRPCBroker Properties and Methods.....	4
2.1.2 TRPCBroker Key Properties.....	5
2.1.3 TRPCBroker Key Methods	6
2.1.4 How to Connect to an M Server	7
2.2 TCCOWRPCBroker Component	9
2.2.1 Single Signon/User Context (SSO/UC).....	9
2.3 TXWBRichEdit Component	9
2.4 TXWBSSOIToken Component	10
3 Remote Procedure Calls (RPCs).....	11
3.1 What is a Remote Procedure Call?	11
3.1.1 Relationship between an M Entry Point and an RPC.....	11
3.2 Create Your Own RPCs.....	11
3.2.1 Preliminary Considerations	11
3.2.2 Process	12
3.3 Writing M Entry Points for RPCs	12
3.3.1 First Input Parameter for RPCs (Required)	12
3.3.2 Return Value Types for RPCs	13
3.3.3 Input Parameter Types for RPCs (Optional).....	15
3.3.4 RPC M Entry Point Examples.....	15
3.4 RPC Entry in the REMOTE PROCEDURE File.....	16
3.5 What Makes a Good Remote Procedure Call?	17
3.6 How to Execute an RPC from a Client Application	17
3.7 RPC Security: How to Register an RPC	19
3.7.1 Bypassing RPC Security for Development	19
3.7.2 BrokerExample Online Code Example	20

4	Other RPC Broker APIs.....	22
4.1	GetServerInfo Function	22
4.1.1	Overview	22
4.1.2	Syntax	23
4.2	VistA Splash Screen Procedures	24
4.3	XWB GET VARIABLE VALUE RPC.....	25
4.4	M Emulation Functions	26
4.4.1	Translate Function.....	26
4.5	Encryption Functions.....	26
4.5.1	In Delphi.....	26
4.5.2	On the VistA M Server	26
4.6	\$\$BROKER^XWBLIB.....	27
4.7	\$\$RTRNFMT^XWBLIB.....	27
5	Broker Security Enhancement (BSE)	28
5.1	Introduction	28
5.1.1	Features	29
5.1.2	Architectural Scope	29
5.2	Process Overview	29
5.2.1	Process Diagrams	33
5.3	BSE-related VistA Applications and Modules.....	35
5.4	Kernel—Authentication Interface to VistA.....	36
5.5	RPC Broker	36
5.5.1	Client	36
5.5.2	Server.....	37
5.6	REMOTE APPLICATION (#8994.5) File	38
5.7	Security Phrase	39
5.8	Kernel Authentication Token	40
6	Debugging and Troubleshooting.....	41
6.1	How to Debug Your Client Application	41
6.1.1	RPC Error Trapping	41
6.2	Troubleshooting Connections	41
6.2.1	Identifying the Listener Process on the Server.....	41
6.2.2	Identifying the Handler Process on the Server	42
6.2.3	Testing Your RPC Broker Connection	42
7	RPC Broker and Delphi.....	43
7.1	Delphi 10.4, 10.3, 10.2, 10.1, 10.0, and XE8 Packages.....	43
7.1.1	Delphi <i>Starter</i> Edition— <i>Not</i> Recommended for BDK Development	43
7.1.2	XWB_RXE#.bpl File	44
7.1.3	XWB_DXE#.bpl File	44

8	RPC Broker Dynamic Link Library (DLL)	45
8.1	DLL Interface	45
8.1.1	Exported Functions	45
8.1.2	Header Files Provided	45
8.1.3	Return Values from RPCs	46
8.1.4	COTS Development and the DLL	46
	Glossary	47
	Index	50

List of Figures

Figure 1:	OnCreate Event Handler—Sample Code	8
Figure 2:	RPC M Entry Point Example—Sum of Two Numbers	15
Figure 3:	RPC M Entry Point Example—Sorted Array	16
Figure 4:	Param Property—Sample Settings	18
Figure 5:	Exception Handler—try...except Code—Sample Usage	18
Figure 6:	RPC Broker Example Application	21
Figure 7:	Server and Port Configuration Selection Dialogue	22
Figure 8:	Sample Registry Information	23
Figure 9:	VistA Splash Screen	24
Figure 10:	Displaying a VistA Splash Screen: Sample Code	25
Figure 11:	XWB GET VARIABLE VALUE RPC Usage—Sample Code	25
Figure 12:	Encryption in VistA M Server—Sample Code	26
Figure 13:	Decryption in VistA M Server—Sample Code	27
Figure 14:	BSE—Process Sequence Flow Diagram	33
Figure 15:	BSE—Process Overview	34

List of Tables

Table 1:	Documentation Symbol Descriptions	xv
Table 2:	Commonly Used RPC Broker Terms	xvii
Table 3:	TRPCBroker Component Key Properties	5
Table 4:	TRPCBroker Component Methods	6
Table 5:	RPC Broker Return Value Types	13
Table 6:	Input Parameter Types	15
Table 7:	REMOTE PROCEDURE File Key Field Entries	16
Table 8:	BSE—Application Authentication Server Class Types	32
Table 9:	BSE—Software Applications and Modules	35

Table 10: Fields in the REMOTE APPLICATION (#8994.5) File	38
Table 11: Header Files that Provide Correct Declarations for DLL Functions	45
Table 12: TRPCBroker Component's Results Property	46
Table 13: Glossary of Terms and Acronyms.....	47

Orientation

How to Use this Manual

Throughout this manual, advice and instructions are offered regarding the use of the Remote Procedure Call (RPC) Broker 1.1 Development Kit (BDK) and the functionality it provides for Veterans Health Information Systems and Technology Architecture (VistA).

Intended Audience

The intended audience of this manual is the following stakeholders:

- Enterprise Program Management Office (EPMO)—VistA legacy development teams.
- System Administrators—System administrators at Department of Veterans Affairs (VA) regional and local sites who are responsible for computer management and system security on the VistA M Servers.
- Information Security Officers (ISOs)—Personnel at VA sites responsible for system security.
- Product Support (PS).

Disclaimers

Software Disclaimer

This software was developed at the Department of Veterans Affairs (VA) by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is *not* subject to copyright protection and is in the public domain. VA assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. We would appreciate acknowledgement if the software is used. This software can be redistributed and/or modified freely provided that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified.



CAUTION: To protect the security of VistA systems, distribution of this software for use on any other computer system by VistA sites is prohibited. All requests for copies of this software for *non-VistA* use should be referred to the VistA site's local Office of Information and Technology Field Office (OITFO).

Documentation Disclaimer

This manual provides an overall explanation of RPC Broker and the functionality contained in RPC Broker 1.1; however, no attempt is made to explain how the overall VistA programming system is integrated and maintained. Such methods and procedures are documented elsewhere. We suggest you look at the various VA Internet and Intranet Websites for a general orientation to VistA. For example, visit the Office of Information and Technology (OIT) VistA Development Intranet website.



DISCLAIMER: The appearance of any external hyperlink references in this manual does *not* constitute endorsement by the Department of Veterans Affairs (VA) of this Website or the information, products, or services contained therein. The VA does *not* exercise any editorial control over the information you find at these locations. Such links are provided and are consistent with the stated purpose of this VA Intranet Service.

Documentation Conventions

This manual uses several methods to highlight different aspects of the material:

- Various symbols are used throughout the documentation to alert the reader to special information. [Table 1](#) gives a description of each of these symbols:

Table 1: Documentation Symbol Descriptions

Symbol	Description
	NOTE / REF: Used to inform the reader of general information including references to additional reading material.
	CAUTION / RECOMMENDATION / DISCLAIMER: Used to caution the reader to take special notice of critical information.

- Descriptive text is presented in a proportional font (as represented by this font).
- Conventions for displaying TEST data in this document are as follows:
 - The first three digits (prefix) of any Social Security Numbers (SSN) begin with either “000” or “666.”
 - Patient and user names are formatted as follows:
 - *[Application Name]*PATIENT,[N]
 - *[Application Name]*USER,[N]

Where “[*Application Name*]” is defined in the Approved Application Abbreviations document and “[*N*]” represents the first name as a number spelled out and incremented with each new entry.

For example, in RPC Broker (XWB) test patient names would be documented as follows:

XWBPATIENT,ONE; XWBPATIENT,TWO; XWBPATIENT,14, etc.

For example, in RPC Broker (XWB) test user names would be documented as follows:

XWBUSER,ONE; XWBUSER,TWO; XWBUSER,14, etc.

- “Snapshots” of computer online displays (i.e., screen captures/dialogues) and computer source code are shown in a *non*-proportional font and may be enclosed within a box.
- User’s responses to online prompts are in **boldface** and highlighted in yellow (e.g., **<Enter>**).
- Emphasis within a dialogue box is in **boldface** and highlighted in blue (e.g., **STANDARD LISTENER: RUNNING**).
- Some software code reserved/key words are in **boldface** with alternate color font.
- References to “<**Enter**>” within these snapshots indicate that the user should press the <**Enter**> key on the keyboard. Other special keys are represented within < > angle brackets. For example, pressing the **PF1** key can be represented as pressing <**PF1**>.
- Author’s comments are displayed in italics or as “callout” boxes.



NOTE: Callout boxes refer to labels or descriptions usually enclosed within a box, which point to specific areas of a displayed image.

- This manual refers to the M programming language. Under the 1995 American National Standards Institute (ANSI) standard, M is the primary name of the MUMPS programming language, and MUMPS is considered an alternate name. This manual uses the name M.
- All uppercase is reserved for the representation of M code, variable names, or the formal name of options, field/file names, and security keys (e.g., the XUPROGMODE security key).



NOTE: Other software code (e.g., Delphi/Pascal and Java) variable names and file/folder names can be written in lower or mixed case.

Documentation Navigation

This document uses Microsoft® Word’s built-in navigation for internal hyperlinks. To add **Back** and **Forward** navigation buttons to your toolbar, do the following:

1. Right-click anywhere on the customizable Toolbar in Word (*not* the Ribbon section).
2. Select **Customize Quick Access Toolbar** from the secondary menu.
3. Press the drop-down arrow in the “Choose commands from:” box.
4. Select **All Commands** from the displayed list.
5. Scroll through the command list in the left column until you see the **Back** command (circle with arrow pointing left).
6. Click/Highlight the **Back** command and press **Add** to add it to your customized toolbar.
7. Scroll through the command list in the left column until you see the **Forward** command (circle with arrow pointing right).
8. Click/Highlight the **Forward** command and press **Add** to add it to your customized toolbar.
9. Press **OK**.

You can now use these **Back** and **Forward** command buttons in your Toolbar to navigate back and forth in your Word document when clicking on hyperlinks within the document.




NOTE: This is a one-time setup and is automatically available in any other Word document once you install it on the Toolbar.

Commonly Used Terms

[Table 2](#) lists terms and their descriptions that can be helpful while reading the RPC Broker documentation:

Table 2: Commonly Used RPC Broker Terms

Term	Description
Client	A single term used interchangeably to refer to a user, the workstation (i.e., PC), and the portion of the program that runs on the workstation.
Component	A software object that contains data and code. A component may or may <i>not</i> be visible.  REF: For a more detailed description, see the <i>Embarcadero Delphi for Windows User Guide</i> .
GUI	The Graphical User Interface application that is developed for the client workstation.

Term	Description
Host	The term Host is used interchangeably with the term Server.
Server	The computer where the data and the RPC Broker remote procedure calls (RPCs) reside.



REF: For additional terms and definitions, see the “[Glossary](#).”

How to Obtain Technical Information Online

Exported VistA M Server-based software file, routine, and global documentation can be generated using Kernel, MailMan, and VA FileMan utilities.



NOTE: Methods of obtaining specific technical information online are indicated where applicable under the appropriate section.

REF: For further information, see the *RPC Broker Technical Manual*.

Help at Prompts

VistA M Server-based software provides online help and commonly used system default prompts. Users are encouraged to enter question marks at any response prompt. At the end of the help display, you are immediately returned to the point from which you started. This is an easy way to learn about any aspect of VistA M Server-based software.

Obtaining Data Dictionary Listings

Technical information about VistA M Server-based files and the fields in files is stored in data dictionaries (DD). You can use the **List File Attributes** [DILIST] option on the **Data Dictionary Utilities** [DI DDU] menu in VA FileMan to print formatted data dictionaries.



REF: For details about obtaining data dictionaries and about the formats available, see the “List File Attributes” chapter in the “File Management” section of the *VA FileMan Advanced User Manual*.

Assumptions

This manual is written with the assumption that the reader is familiar with the following:

- VistA computing environment:
 - Kernel—VistA M Server software
 - Remote Procedure Call (RPC) Broker—VistA Client/Server software
 - VA FileMan data structures and terminology—VistA M Server software
- Microsoft Windows environment
- M programming language
- Object Pascal programming language
- Object Pascal programming language/Embarcadero Delphi Integrated Development Environment (IDE)—RPC Broker

References

Readers who wish to learn more about RPC Broker should consult the following:

- *RPC Broker Release Notes*
- *RPC Broker Deployment, Installation, Back-Out, and Rollback Guide (DIBRG)*
- *RPC Broker Systems Management Guide*
- *RPC Broker Technical Manual*
- *RPC Broker User Guide* (this manual)
- *RPC Broker Developer's Guide*
- RPC Broker VA Intranet website.

This site provides announcements, additional information (e.g., Frequently Asked Questions [FAQs], advisories), documentation links, archives of older documentation and software downloads.

VistA documentation is made available online in Microsoft® Word format and in Adobe Acrobat Portable Document Format (PDF). The PDF documents *must* be read using the Adobe Acrobat Reader, which is freely distributed by Adobe Systems Incorporated at: <http://www.adobe.com/>

VistA documentation can be downloaded from the VA Software Document Library (VDL) Website: <http://www.va.gov/vdl/>

RPC Broker documentation is located on the VDL at:
<https://www.va.gov/vdl/application.asp?appid=23>

VistA documentation and software can also be downloaded from the Product Support (PS) Anonymous Directories.

1 Introduction

The Remote Procedure Call (RPC) Broker (also referred to as “Broker”) is a client/server system within Department of Veterans Affairs (VA) Veterans Health Information Systems and Technology Architecture (VistA) environment. It establishes a common and consistent foundation for client/server applications being written as part of VistA. It enables client applications to communicate and exchange data with M Servers.

This manual provides an overview of software development with the RPC Broker. It introduces developers to the RPC Broker and the Broker Development Kit (BDK) with emphasis on using the RPC Broker in conjunction with Embarcadero’s Delphi software. However, the RPC Broker supports other development environments.



REF: For more complete information on development with the RPC Broker components, see the *RPC Broker Developer’s Guide*.

This document is intended for the VistA development community and system administrators. A wider audience of technical personnel engaged in operating and maintaining the Department of Veterans Affairs (VA) software can also find it useful as a reference.

1.1 About this Version of the BDK

RPC Broker 1.1 (fully patched) provides developers with the capability to create new VistA client/server software using the following RPC Broker Delphi components in the 32-bit environment:

- **TCCOWRPCBroker**
- **TContextorControl**
- **TRPCBroker** (original component)
- **TXWBRichEdit**
- **TXWBSSOi**



NOTE: These RPC Broker components wrap the functionality of the Broker resulting in a more modularized and orderly interface. Those components derived from the original **TRPCBroker** component, inherit the **TRPCBroker** properties and methods.

1.1.1 Features

This enhanced Broker software has the following functionality/features:

- Selection of the User's Authentication Certificate—Eliminates the need for the user to select from a list of certificates.
- Supports Active Directory (AD) Credentials—When a user is unable to log onto a workstation with their Personal Identity Verification (PIV) card, the user contacts the Enterprise Service Desk (ESD) to receive a PIV exemption to allow them to log on with their Active Directory (AD) credentials (username and password). This enhanced BDK detects this condition and allows the user to use their AD credentials to secure a SAML token from IAM for logging onto VistA via applications compiled with this version of the BDK.
- Supports 2-factor Authentication—The **TRPCBroker** component authenticates a user by making a mutual Transport Layer Security (TLS) authentication connection to the Identity and Access Management (IAM) Secure Token Service (STS). Mutual authentication refers to two parties authenticating each other at the same time. Mutual TLS authentication uses the TLS protocol to authenticate and identify a user using Public Key Encryption (PKI) certificates (usually found on a portable smart card or device) and a private Personal Identification Number (PIN) to unlock the certificate. The STS server returns a digitally-signed token containing the user's identity. This token is trusted by the VistA M Server as a delegated form of user authentication.
- Supports IPv4/IPv6 Dual-Stack Environment—The **TRPCBroker** component uses WinSock 2.2 Application Programming Interfaces (APIs) that support network connections using Internet Protocol (IP) version 4 and/or IP version 6. IPv6 is a protocol designed to handle the growth rate of the Internet and to cope with the demanding requirements of services, mobility, and end-to-end security.
- Supports Secure Shell (SSH)—The **TRPCBroker** component enabled Secure Shell (SSH) Tunnels to be used for secure connections. This functionality is controlled by setting an internal property value (mandatory SSH) or command line option at run-time.
- Supports Broker Security Enhancement (BSE)—The **TRPCBroker** component enabled visitor access to remote sites using authentication established at a home site.
- Supports Single Sign-On/User context (SSO/UC)—**TCCOWRPCBroker** component enables Single Sign-On/User Context (SSO/UC) in CCOW-enabled applications.
- Supports Non-Callback Connections—By default the RPC Broker components are built with a UCX or *non*-callback Broker connection, so that it can be used from behind firewalls, routers, etc.
- Supports Silent Logon capabilities—RPC Broker provides “Silent Login” capability. It provides functionality associated with the ability to make logins to a VistA M Server without the RPC Broker asking for Access and Verify code information.
- Documented Deferred RPCs and Capability to Run RPCs on a Remote Server.

- Multi-instances of the RPC Broker—RPC Broker code permits an application to open two separate Broker instances with the same Server/ListenerPort combination, resulting in two separate partitions on the server. Previously, an attempt to open a second Broker instance ended up using the same partition. For this capability to be useful for concurrent processing, an application would have to use threads to handle the separate Broker sessions.



CAUTION: Although we believe there should be no problems, the RPC Broker is *not* guaranteed to be thread safe.

- Updated components, properties, methods, and types.
- Separate **DesignTime** and **RunTime** Packages—BDK contains separate **RunTime** and **DesignTime** packages.
- Supports Delphi 10.4, 10.3, 10.2, 10.1, 10, 10.0, and XE8.

To develop Vista applications in a **32**-bit environment you *must* have Delphi XE8 or greater. However, the Broker routines on the M server continue to support Vista applications previously developed in the **16**-bit environment.

The default installation of the Broker creates a separate BDK directory (i.e., BDK32) that contains the required Broker files for development.



REF: For a complete list of all new or modified features and functionality with RPC Broker 1.1, see the *RPC Broker Release Notes*.

1.1.2 Backward Compatibility Issues

Client applications compiled with RPC Broker 1.1 will *not* work at a site that has *not* upgraded its RPC Broker server software to Version 1.1.

On the other hand, client applications compiled with RPC Broker 1.0 will work with the RPC Broker 1.1 server.

2 RPC Broker Components for Delphi



REF: For more detailed information on the RPC Broker components for Delphi, see the *RPC Broker Developer's Guide*.

2.1 TRPCBroker Component

The main tool to develop client applications for the RPC Broker environment is the **TRPCBroker** component for Delphi. The **TRPCBroker** component adds the following abilities to your Delphi application:

- Connecting to an M server:
 - Authenticate the user
 - Set up the environment on the server
 - Bring back the introductory text
- Invoking Remote Procedure Calls (RPCs) on the M Server:
 - Send data to the M Server
 - Perform actions on the server
 - Return data from the server to the client

To add the **TRPCBroker** component to your Delphi application, simply drop it from the Kernel tab of Delphi's component palette to a form in your application.

2.1.1 TRPCBroker Properties and Methods

As a Delphi component, the **TRPCBroker** component is controlled and accessed through its properties and methods. By setting its properties and executing its methods, you can connect to an M server from your application and execute RPCs on the M server to exchange data and perform actions on the M server.

For most applications, you only need to use a single **TRPCBroker** component to manage communications with the M server.

2.1.2 TRPCBroker Key Properties

[Table 3](#) lists the most important properties of the **TRPCBroker** component.



REF: For a complete list of all of Broker properties, see the *RPC Broker Developer's Guide*.

Table 3: TRPCBroker Component Key Properties

Property	Description
ClearParameters	If True , the Param property is cleared <i>after</i> every invocation of the Call , strCall , or the IstCall methods.
ClearResults	If True , the Results property is cleared <i>before</i> every invocation of the Call method, thus assuring that only the results of the last call are returned.
Connected	Setting this property to True connects your application to the server.
ListenerPort	Sets server port to connect to a Broker Listener process (mainly for development purposes; for end-users, determine on the fly with GetServerInfo method.)
Param	RunTime array in which you set any parameters to pass as input parameters when calling an RPC on the server.
RemoteProcedure	Name of a RemoteProcedure entry that the Call , IstCall , or strCall method should invoke.
Results	This is where any results are stored after a Call , IstCall , or strCall method completes.
Server	Name of the server to connect to (mainly for development purposes; for end-users, determine on-the-fly with GetServerInfo method.)
SSHPort	Holds a specific port number for Secure Shell (SSH) Tunneling if the UseSecureConnection property is set to " SSH " or " PLINK ". If <i>not</i> specified, uses the RPC Broker listener port for the remote server.
SSHPw	Holds a password for SSH Tunneling if the UseSecureConnection property is set to " PLINK ".
SSHUser	Holds a specific username for SSH Tunneling if the UseSecureConnection property is set to " SSH ". For VA Vista servers, the username is typically of the form xxxvista where the xxx is the station's three letter abbreviation.
UseSecureConnection	Used to specify whether SSH Tunneling is to be used when making the connection.

2.1.3 TRPCBroker Key Methods

This section lists the most important methods of the **TRPCBroker** component.



REF: For a complete list of all of Broker methods, see the *RPC Broker Developer's Guide*.

Table 4: TRPCBroker Component Methods

Method	Description
procedure Call;	<p>This method executes an RPC on the server and returns the results in the TRPCBroker component's Results property.</p> <p>Call expects the name of the remote procedure and its parameters to be set up in the RemoteProcedure and Param properties respectively.</p> <p>If ClearResults is True, then the Results property is cleared before the call.</p> <p>If ClearParameters is True, then the Param property is cleared after the call finishes.</p>
function strCall: string;	<p>This method is a variation of the Call method. Only use it when the return type is a single string. Instead of returning results in the TRPCBroker component's Results[0] property node, results are returned as the value of the function call. Unlike the Call method, the Results property is <i>not</i> affected; no matter the setting of ClearResults, the value is left unchanged.</p>
procedure lstCall(OutputBuffer: TStrings);	<p>This method is a variation of the Call method. Instead of returning results in the TRPCBroker component's Results property, it instead returns results in the TStrings object you specify. Unlike the Call method, the Results property is <i>not</i> affected; no matter the setting of ClearResults, the value is left unchanged.</p>
function CreateContext(strContext: string): boolean;	<p>This method creates a context for your application. Pass an option name in the strContext parameter. If the function returns True, a context was created, and your application can use all RPCs entered in the option's RPC multiple.</p>

Examples

For examples of how to use these methods to invoke RPCs, see the "[How to Execute an RPC from a Client Application](#)" section.

2.1.4 How to Connect to an M Server

To establish a connection from your application to a Broker server, perform the following procedure:

1. From the **Kernel** component palette tab, add a **TRPCBroker** component to your form.
2. Add code to your application to connect to the server; one likely location is your form's **OnCreate** event handler. The code should:
 - a. Use the **GetServerInfo** function to retrieve the **RunTime** server and port to connect to, and **SSHUsername** if available.



NOTE: This function is *not* a method of the **TRPCBroker** component; it is described in the “[Other RPC Broker APIs](#)” section.

- b. Inside of an exception handler **try...except** block, set **RPCBroker1**'s **Connected** property to **True**. This causes an attempt to connect to the Broker server and authenticates the user.
 - c. Check if an **EBrokerError** exception is raised. If this happens, connection failed. You should inform the user of this and then terminate the application.

The code, placed in an **OnCreate** event handler, should look like [Figure 1](#):

Figure 1: OnCreate Event Handler—Sample Code

```
procedure TForm1.FormCreate(Sender: TObject);
var   ServerStr: String;
      PortStr: String;
begin
    // get the correct port and server from registry
    if GetServerInfo(ServerStr,PortStr,SSHUsernameStr) <> mrCancel then
    begin
        RPCBroker1.Server:=ServerStr;
        RPCBroker1.ListenerPort:=StrToInt(PortStr);
        if SSHUsernameStr <> '' then
        begin
            RPCBroker1.UseSecureConnection := 'SSH';
            RPCBroker1.SSHport := '';
            RPCBroker1.SSHUser := SSHUsernameStr;
            RPCBroker1.SSHpw := '';
            RPCBroker1.SSHHide := true;
        end;
    end
    else Application.Terminate;

    // establish a connection to the Broker
    try
        RPCBroker1.Connected:=True;
    except
        On EBrokerError do
        begin
            ShowMessage('Connection to server could not be established!');
            Application.Terminate;
        end;
    end;
end;
```

3. A connection with the Broker M Server is now established. You can use the **CreateContext** method of the **TRPCBroker** component to authorize use of RPCs for your user, and then use the **Call**, **lstCall**, and **strCall** methods of the **TRPCBroker** component to execute RPCs on the M server.



REF: For information on creating and executing RPCs, see the “[Remote Procedure Calls \(RPCs\)](#)” section.

2.2 TCCOWRPCBroker Component

As of Patch XWB*1.1*40, the **TCCOWRPCBroker** component was added to Version 1.1 of the RPC Broker. The **TCCOWRPCBroker** Delphi component allows VistA application developers to make their applications CCOW-enabled and Single Sign-On/User Context (SSO/UC)-aware with all of the client/server-related functionality in one integrated component. Using the **TCCOWRPCBroker** component, an application can share User Context stored in the CCOW Context Vault.

Thus, when a VistA CCOW-enabled application is recompiled with the **TCCOWRPCBroker** component and other required code modifications are made, that application would then become SSO/UC-aware and capable of single sign-on (SSO).



NOTE: This RPC Broker component is derived from the original [TRPCBroker Component](#); it inherits the **TRPCBroker** properties and methods.

2.2.1 Single Signon/User Context (SSO/UC)

The Veterans Health Administration (VHA) information systems user community expressed a need for a single sign-on (SSO) service with interfaces to VistA, HealtheVet VistA, and *non*-VistA systems. This architecture allows users to authenticate and sign on to multiple applications that are CCOW-enabled and SSO/UC-aware using a single set of credentials, which reduces the need for multiple ID's and passwords in the HealtheVet clinician desktop environment. The RPC Broker software addressed this architectural need by providing a new **TCCOWRPCBroker** component in RPC Broker Patch XWB*1.1*40.

The **TCCOWRPCBroker** component allows VistA application developers to make their applications CCOW-enabled and Single Sign-On/User Context (SSO/UC)-aware with all of the client/server-related functionality in one integrated component. Using the **TCCOWRPCBroker** component, an application can share User Context stored in the CCOW Context Vault.

Thus, when a VistA CCOW-enabled application is recompiled with the **TCCOWRPCBroker** component and other required code modifications are made, that application would then become SSO/UC-aware and capable of single sign-on (SSO).



REF: For more information on SSO/UC and making your Broker-based applications CCOW-enabled and SSO/UC-aware, please consult the *Single Sign-On/User Context (SSO/UC) Installation Guide* and *Single Sign-On/User Context (SSO/UC) Deployment Guide* on the VA Software Document Library (VDL) at:
<https://www.va.gov/vdl/application.asp?appid=162>

2.3 TXWBRichEdit Component

As of Patch XWB*1.1*13, the **TXWBRichEdit** component was added to Version 1.1 of the RPC Broker. The **TXWBRichEdit** Delphi component replaces the Introductory Text Memo component on the Login Form. **TXWBRichEdit** is a version of the **TRichEdit** component that uses Version 2 of Microsoft's **RichEdit** Control and adds the ability to detect and respond to a

Uniform Resource Locator (URL) in the text. This component permits us to provide some requested functionality on the login form. As an **XWB** namespaced component, it *must* be put it on the **Kernel** tab of the component palette, however, it rightly belongs on the **Win32** tab.

2.4 TXWBSSOiToken Component

As of Patch XWB*1.1*65, the **TXWBSSOiToken** component was added to RPC Broker 1.1. The **TXWBSSOiToken** Delphi component is used to authenticate a user into the Identity and Access Management (IAM) Secure Token Service (STS) and obtain a Security Assertion Markup Language (SAML) token containing an authenticated user's identity. The **TXWBSSOiToken** component does *not* need to be specifically added to an RPC Broker application, as authentication is built into the [TRPCBroker Component](#). However, it is made available as a separate component for those applications that might need to obtain a SAML token for authentication into *non*-RPC Broker applications or servers.

3 Remote Procedure Calls (RPCs)

3.1 What is a Remote Procedure Call?

A remote procedure call (RPC) is a defined call to M code that runs on an M server. A client application, through the RPC Broker, can make a call to the M server and execute an RPC on the M server. This is the mechanism through which a client application can:

- Send data to an M server.
- Execute code on an M server.
- Retrieve data from an M server.

An RPC can take optional parameters to do some task and then return either a single value or an array to the client application. RPCs are stored in the REMOTE PROCEDURE (#8994) file.

3.1.1 Relationship between an M Entry Point and an RPC

An RPC can be thought of as a wrapper placed around an M entry point for use with client applications. Each RPC invokes a single M entry point. The RPC passes data in specific ways to its corresponding M entry point and expects any return values from the M entry point to be returned in a pre-determined format. This allows client applications to connect to the RPC Broker, invoke an RPC, and through the RPC, invoke an M entry point on a server.

3.2 Create Your Own RPCs

3.2.1 Preliminary Considerations

Because creating a Remote Procedure Call (RPC) could introduce security risks, you should consider your options prior to creating a new one:

1. First, look for an existing RPC that provides the data you need. You may need an Integration Control Registration (ICR) for permission to use the RPC.
2. If you *cannot* locate an existing RPC that meets your needs, look for an existing Application Programming Interface (API) that can be wrapped with a new RPC.
3. If an existing RPC or API provides “almost” what you need, contact the package owners to see whether there is a modification or alternative that could be provided to meet your needs. For example, determine whether post-processing of the data in your application would provide the results you need.
4. You should create a new RPC only as a last result. When creating a new RPC is necessary, you should carefully consider how general to make the RPC, so that it can potentially be used by other applications in the future.

3.2.2 Process

You can create your own custom RPCs to perform actions on the M server and to retrieve data from the M server. Then you can call these RPCs from your client application. Creating an RPC requires you to perform the following steps:

1. Reference the [RPC Broker Developers Guide](#) for instructions and examples when creating a new RPC.
2. Write and test the M entry point that is called by the RPC.
3. Add the RPC entry that invokes your M entry point, in the REMOTE PROCEDURE (#8994) file. The RPC name should begin with the VistA package namespace that owns the RPC. For example, “XWB EXAMPLE BIG TEXT” is owned by the RPC Broker package (namespace: XWB). M Programming Standards and Conventions (SAC) provide policy on name requirements for new RPCs.
4. Add the RPC to a “B-Broker (Client/Server)” type option in the OPTION (#19) file. The option should be in your VistA package namespace. M Programming Standards and Conventions (SAC) provide policy on name requirements for options.

3.3 Writing M Entry Points for RPCs

3.3.1 First Input Parameter for RPCs (Required)


The RPC Broker always passes a variable by reference in the first input parameter to your M routine. It expects results (one of five types described in [Table 5](#)) to be returned in this parameter. You *must* always set some return value into that first parameter before your routine returns.

3.3.2 Return Value Types for RPCs

[Table 5](#) lists the **five** RETURN VALUE TYPES for RPCs. Choose a return value type that is appropriate to the type of data your RPC needs to return to your client. Your M entry point should set the return value (in the routine's first input parameter) accordingly.

Table 5: RPC Broker Return Value Types

RPC Return Value Type	How M Entry Point Should Set the Return Parameter	RPC WORD WRAP ON Setting	Value(s) returned in Client Results
Single Value	Set the return parameter to a single value. For example: <pre>TAG (RESULT) ; S RESULT="DOE, JOHN" Q</pre>	No effect	Value of parameter, in Results[0] .
Array	Set an array of strings into the return parameter, each subscripted one level descendant. For example: <pre>TAG (RESULT) ; S RESULT (1) ="ONE" S RESULT (2) ="TWO" Q</pre> <p>For large arrays consider using the GLOBAL ARRAY return value type to avoid memory allocation errors.</p>	No effect	Array values, each in a Results item.
Word-processing	Set the return parameter the same as you set it for the Array type. The only difference is that the WORD WRAP ON (#.08) field setting affects the Word-Processing return value type.	True	Array values, each in a Results item.
		False	Array values concatenated into Results[0] .
Global Array	Set the return parameter to a closed global reference in ^TMP . The global's data nodes are traversed using \$QUERY , and all data values on global nodes descendant from the global reference are returned. The Global Array type is especially useful for returning data from VA FileMan word-processing fields, where each line is on a 0 -subscripted node.	True	Array values, each in a Results item.
		False	Array values concatenated into Results[0] .


RPC Return Value Type	How M Entry Point Should Set the Return Parameter	RPC WORD WRAP ON Setting	Value(s) returned in Client Results
	 <p>CAUTION: The global reference you pass is killed by the Broker at the end of RPC Execution as part of RPC cleanup. Do not pass a global reference that is <i>not</i> in ^TMP or that should <i>not</i> be killed.</p> <p>This type is useful for returning large amounts of data to the client, where using the Array type can exceed the symbol table limit and crash your RPC. For example, to return signon introductory text you could do:</p> <pre> TAG (RESULT) ; M ^TMP ("A6A", \$J) = ^XTV (8989.3, 1, "INTRO") ;this node not needed K ^TMP ("A6A", \$J, 0) S RESULT=\$NA (^TMP ("A6A", \$J)) Q </pre>		
Global Instance	<p>Set the return parameter to a closed global reference.</p> <p>For example, to return the 0th node from the NEW PERSON (#200) file for the current user:</p> <pre> TAG (RESULT) ; S RESULT=\$NA (^VA (200, DUZ, 0)) Q </pre>	No effect	Value of global node, in Results[0] .

3.3.3 Input Parameter Types for RPCs (Optional)

The M entry point for an RPC can optionally have input parameters (i.e., beyond the first parameter, which is always used to return an output value). The client passes data to your M entry point through these parameters.

[Table 6](#) lists the three format types that the client can send data to an RPC (and therefore your entry point):

Table 6: Input Parameter Types

Param PType	Param Value
Literal	Delphi string value; passed as a string literal to the M server.
Reference	<div>Delphi string value; treated on the M Server as an M variable name and resolved from the symbol table at the time the RPC executes.</div> <div> CAUTION: For enhanced security reasons, the reference parameter type may be deprecated and removed in subsequent updates to the BDK.</div>
List	A single-dimensional array of strings in the Mult subproperty of the Param property, passed to the M Server where it is placed in an array. String subscripting can be used.

The type of the input parameters passed in the **Param** property of the **TRPCBroker** component determines the format of the data you *must* be prepared to receive in your M entry point.

3.3.4 RPC M Entry Point Examples

The following two examples illustrate sample M code that could be used in simple RPCs.

3.3.4.1 Sum of Two Numbers

The example in [Figure 2](#) takes two numbers and returns their sum:

Figure 2: RPC M Entry Point Example—Sum of Two Numbers

```
SUM(RESULT,A,B)    ;add two numbers
S RESULT=A+B
Q
```

3.3.4.2 Sorted Array

The example in [Figure 3](#) receives an array of numbers and returns them as a sorted array to the client:

Figure 3: RPC M Entry Point Example—Sorted Array


```
SORT (RESULT, UNSORTED)      ;sort numbers
N I
S I=""
F S I=$O(UNSORTED(I)) Q:I="" S RESULT(UNSORTED(I))=UNSORTED(I)
Q
```

3.4 RPC Entry in the REMOTE PROCEDURE File

After the M code is complete, you need to create the RPC itself in the REMOTE PROCEDURE (#8994) file. The following fields in the REMOTE PROCEDURE (#8994) file are key to the correct operation of an RPC:

Table 7: REMOTE PROCEDURE File Key Field Entries

Field Name	Required?	Description
NAME (#.01)	Yes	The name that identifies the RPC (this entry should be namespaced in the package namespace).
TAG (#.02)	Yes	The tag at which the remote procedure call begins.
ROUTINE (#.03))	Yes	The name of the routine that should be invoked to start the RPC.
WORD WRAP ON (#.08)	No	Affects Global Array and Word-Processing return value types only: <ul style="list-style-type: none">• If set to False, data is returned in a single concatenated string in Results[0].• If set to True, each array node on the M side is returned as a distinct array item in Results.
RETURN VALUE TYPE (#.04)	Yes	This indicates to the Broker how to format the return values. For example, if the RETURN VALUE TYPE is set as Word-Processing , then each entry in the returning list has a <CR><LF> (<carriage return><line feed>) appended.
APP PROXY ALLOWED (#.11)	No	This field <i>must</i> be set to Allowed (1) if this RPC is to be run by an APPLICATION PROXY user. The default is to <i>not</i> allow access.

Field Name	Required?	Description
		 CAUTION: APPLICATION PROXY users do <i>not</i> meet Health Insurance Portability and Accounting Act of 1996 (HIPAA) requirements for user identification and should <i>not</i> be permitted to access an RPC that reads or writes Personal Health Information (PHI).
PARAMETER TYPE (#8994.02,.02) field of the INPUT PARAMETER Multiple (#8994.02)	Yes	<p>This field is used to indicate the type (Literal, List, Reference, or Word-Processing entry) of value passed by this parameter. The Literal, List, and Reference types correspond to the TParamType of the same name. A Word-Processing type would also be a List TParamType.</p> <p>Currently, this declaration is mandatory for a reference, but <i>recommended</i> for other types. In the future the parameter type declaration will be enforced for all types.</p>

3.5 What Makes a Good Remote Procedure Call?

The following make for a good remote procedure call (RPC) :

- Silent calls (no I/O to terminal or screen, no user intervention required).
- Minimal resources required (passes data in brief, controlled increments).
- Discrete calls (requiring as little information as possible from the process environment).
- Generic as possible (different parts of the same package as well as other packages could use the same RPC).

3.6 How to Execute an RPC from a Client Application

To execute an RPC from a client application, perform the following procedure:

1. If your RPC has any input parameters beyond the mandatory first parameter, set a **Param** node in the **TRPCBroker**'s **Param** property for each. For each input parameter, set the following sub properties:
 - **Value**
 - **PType** (**Literal**, **List**, or **Reference**)

If the parameter's **PType** is **List**, however, set a list of values in the **Mult** subproperty rather than setting the **Value** subproperty.



CAUTION: For enhanced security reasons, the reference parameter type may be deprecated and removed in subsequent updates to the BDk.

[Figure 4](#) is an example of some settings of the **Param** property:

Figure 4: Param Property—Sample Settings

```
RPCBroker1.Param[0].Value := '10/31/97';  
RPCBroker1.Param[0].PType := literal;  
RPCBroker1.Param[1].Mult['NAME'] := 'XWBUSER, ONE';  
RPCBroker1.Param[1].Mult['SSN'] := '000-45-6789';  
RPCBroker1.Param[1].PType := list;
```

2. Set the **TRPCBroker**'s **RemoteProcedure** property to the name of the RPC to execute.

```
RPCBroker1.RemoteProcedure := 'A6A LIST';
```

3. Invoke the **Call** method of the **TRPCBroker** component to execute the RPC. All calls to the **Call** method should be done within an exception handler **try...except** statement, so that all communication errors (which trigger the **EBrokerError** exception) can be trapped and handled. For example:

Figure 5: Exception Handler—try...except Code—Sample Usage

```
try  
    RPCBroker1.Call;  
except  
    On EBrokerError do  
        ShowMessage('A problem was encountered communicating with the server.');
```

4. Any results returned by your RPC are returned in the **TRPCBroker** component's **Results** property. Depending on how you set up your RPC, results are returned either in a single node of the **Results** property (**Result[0]**) or in multiple nodes of the **Results** property.



NOTE: You can also use the **lstCall** and **strCall** methods to execute an RPC. The main difference between these methods and the **Call** method is that **lstCall** and **strCall** do *not* use the **Results** property, instead returning results into a location you specify.

3.7 RPC Security: How to Register an RPC

Security for RPCs is handled through the RPC registration process. Each client application *must* create a context for itself, which checks if the application user has access to a “**B**”-type option in the Kernel menu system. Only RPCs assigned to that option can be run by the client application.

To enable your application to create a context for itself, perform the following procedure:

1. Create a “**B**”-type option in the OPTION (#19) file for your application.



NOTE: The OPTION TYPE “**B**” represents a **Broker** client/server type option.

2. In the RPC multiple for this option type, add an entry for each RPC that your application calls. You can also specify a security key that can lock each RPC (this is a pointer to the SECURITY KEY [#19.1] file) and M code in the RULES subfield that can also determine whether to enable access to each RPC.
3. When you export your software using KIDS, export both your RPCs and your software option.
4. Your application *must* create a context for itself on the server, which checks access to RPCs. In the initial code of your client application, make a call to the **CreateContext** method of your **TRPCBroker** component. Pass your application’s “**B**”-type option’s name as a parameter. For example:

```
RPCBroker1.CreateContext(option_name)
```

If the **CreateContext** method returns **True**, only those RPCs designated in the RPC multiple of your application option are permitted to run.

If the **CreateContext** method returns **False**, you should terminate your application (if you do *not*, your application runs, but you get errors every time you try to access an RPC).

5. End-users of your application *must* have the “**B**”-type option assigned to them on one of their menus, in order for the **CreateContext** method to return **True**.

3.7.1 Bypassing RPC Security for Development

Having the XUPROGMODE security key allows you to bypass the Broker security checks. You can run any RPC without regard to application context (without having to use the **CreateContext** method). This is a convenience for application development. When you complete development, make sure you test your application from an account *without* the XUPROGMODE key, to ensure that all RPCs needed are properly registered.

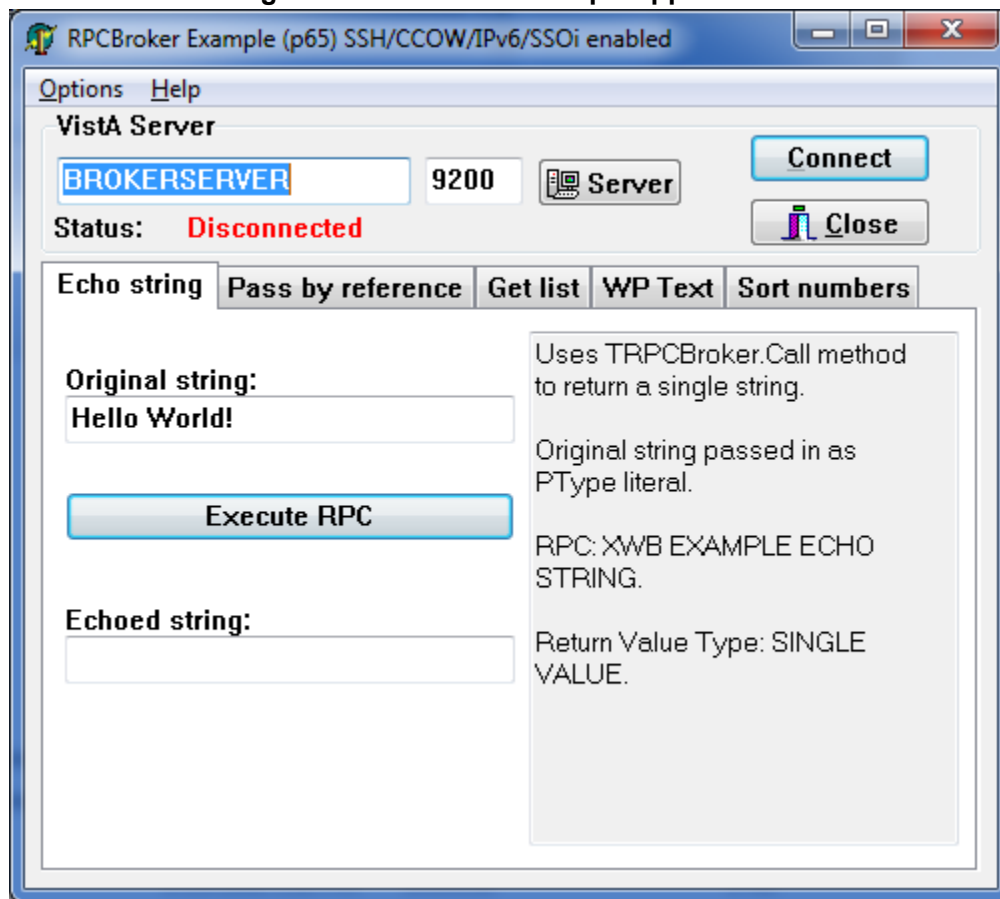
3.7.2 BrokerExample Online Code Example

The BrokerExample sample application (i.e., **BROKEREXAMPLE.EXE**) provided with the BDK demonstrates the basic features of developing RPC Broker-based applications, including:

- Connecting to an M server.
- Creating an application context.
- Using the **GetServerInfo** function.
- Displaying the VistA splash screen.
- Setting the **TRPCBroker**'s **Param** property for each Param **PType** (**literal**, **reference**, and **list**).
- Calling RPCs with the **Call** method.
- Calling RPCs with the **lstCall** and **strCall** methods.
- Secure Shell (SSH) connection (from **Options** menu) methods.

The client source code files for the **BrokerExample** application are located in the **SAMPLES\RPCBROKER\BROKEREX** subdirectory of the main **BDK32** directory.

Figure 6: RPC Broker Example Application



4 Other RPC Broker APIs

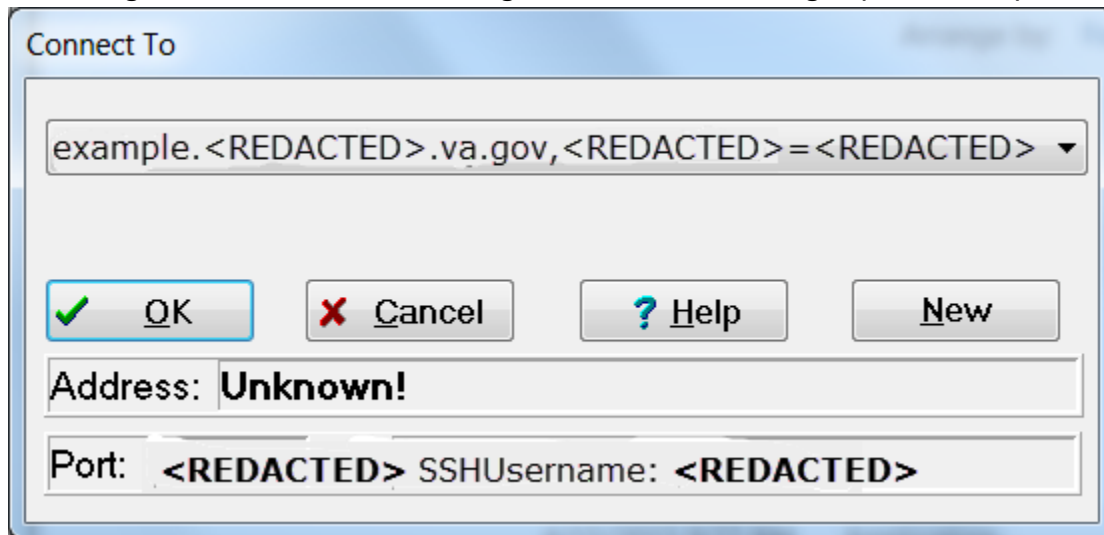
4.1 GetServerInfo Function

4.1.1 Overview

The **GetServerInfo** function retrieves the end-user workstation's server, port, and **SSHUsername** if available. Use this function to set the **TRPCBroker** component's **Server**, **ListenerPort**, and **SSHUser** properties to reflect the end-user workstation's settings before connecting to the server.

If there is more than one server/port to choose from, **GetServerInfo** displays dialogue that allows users to select a service to connect to, as shown in [Figure 7](#):

Figure 7: Server and Port Configuration Selection Dialogue (REDACTED)



If exactly one server and port entry is defined in the Microsoft Windows Registry, **GetServerInfo** does *not* display the dialogue in [Figure 7](#). The values in the single Microsoft Windows Registry entry are returned, with no user interaction required.

If more than one server and port entry exists in the Microsoft® Windows Registry, the dialogue is displayed, and the user chooses to which server they want to connect.

If no values for server and port are defined in the Microsoft® Windows Registry, **GetServerInfo** does *not* display the dialogue in [Figure 7](#), and automatic default values are returned (i.e., **BROKERSERVER** and **<REDACTED>**).

The values are stored in either of the following registries:

- **HKEY_CURRENT_USER (HKCU)**
- **HKEY_LOCAL_MACHINE (HKLM)**

These registries are located under:

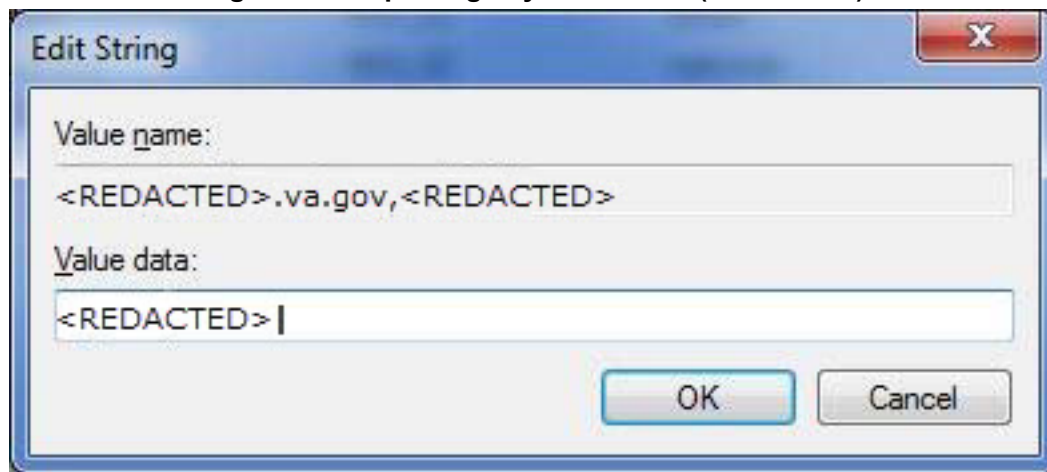
\Software\Vista\Broker\Servers

Entries are of the format:

- Name: **Server,ListenerPort**
- Type: **REG_SZ**
- Data: **SSHUser**

For example, a connection to server address “<REDACTED>.va.gov” using port <REDACTED> and SSHUsername of “<REDACTED>” would look like:

Figure 8: Sample Registry Information (REDACTED)



4.1.2 Syntax

Two versions of the **GetServerInfo** function are supported:

- **Legacy Version**—Retrieves the end user’s server and port:

```
function GetServerInfo(var Server, Port: string): integer;
```
- **New Version**—Retrieves the end user’s server and port as well as the SSHUsername value from the Windows registry:

```
function GetServerInfo(var Server, Port, SSHUsername: string): integer;
```

Both versions continue to support specification of SSHUsername at the command line.



NOTE: The unit is **RpcConf1**.

4.2 VistA Splash Screen Procedures

Two procedures in **SplVista.PAS** unit are provided to display a VistA splash screen when an application loads:

- **procedure** `SplashOpen;`
- **procedure** `SplashClose (TimeOut : longint);`

It is *recommended* that the splash screen be opened and closed in the section of Pascal code in an application's project file (i.e., **.DPR**).

To use the splash screen in an application, perform the following procedure:

1. Open your application's project (**.DPR**) file (in Delphi, choose **View | Project Source**).
2. Include the **SplVista** in the uses clause of the project source.
3. Call **SplashOpen** immediately after the first form of your application is created and call **SplashClose** just prior to invoking the **Application.Run** method.
4. Use the **TimeOut** parameter to ensure a minimum display time.

Figure 9: VistA Splash Screen



Figure 10: Displaying a Vista Splash Screen: Sample Code

```
uses
    Forms, Unit1 in 'Unit1.pas', SplVista;

{$R *.RES}

begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    SplashOpen;
    SplashClose(2000);
    Application.Run;
end.
```

4.3 XWB GET VARIABLE VALUE RPC

You can call the **XWB GET VARIABLE VALUE** RPC (distributed with the RPC Broker) to retrieve the value of any M variable in the server environment. Pass the variable name in **Param[0].Value** and the type (reference) in **Param[0].PType**. Also, the current context of your user *must* give them permission to execute the **XWB GET VARIABLE VALUE** RPC (it *must* be included in the RPC multiple of the “**B**”-type option registered with the **CreateContext** function). For example:

Figure 11: XWB GET VARIABLE VALUE RPC Usage—Sample Code

```
RPCBroker1.RemoteProcedure := 'XWB GET VARIABLE VALUE';
RPCBroker1.Param[0].Value := 'DUZ';
RPCBroker1.Param[0].PType := reference;
try
    RPCBroker1.Call;
except
    On EBrokerError do
        ShowMessage('Connection to server could not be established!');
end;
ShowMessage('DUZ is '+RPCBroker1.Results[0]);
```



CAUTION: For enhanced security reasons, the reference parameter type may be deprecated and removed in subsequent updates to the BDK.

4.4 M Emulation Functions

Piece Function

The **Piece** function is a scaled down Pascal version of M's **\$PIECE** function. It is declared in **MFUNSTR.PAS**.

```
function Piece(x: string; del: string; piece: integer) : string;
```

4.4.1 Translate Function

The **Translate** function is a scaled down Pascal version of M's **\$TRANSLATE** function. It is declared in **MFUNSTR.PAS**.

```
function Translate(passedString, identifier, associator: string): string;
```

4.5 Encryption Functions

Kernel and the RPC Broker provide some rudimentary encryption and decryption functions. Data can be encrypted on the client end and decrypted on the server, and vice-versa.

4.5.1 In Delphi

Include **HASH** in the “uses” clause of the unit in which you’ll be encrypting or decrypting.

Function prototypes are as follows:

- `function Decrypt(EncryptedText: string): string;`
- `function Encrypt(NormalText: string): string;`

4.5.2 On the VistA M Server

4.5.2.1 Encryption

To encrypt:

Figure 12: Encryption in VistA M Server—Sample Code

```
>S CIPHER=$$ENCRYP^XUSRB1("Hello world!") W CIPHER  
/U'11TG~TV1&f-
```

4.5.2.2 Decryption

To decrypt:

Figure 13: Decryption in VistA M Server—Sample Code

```
>S PLAIN=$$DECRYPT^XUSRB1(CIPHER) W PLAIN  
Hello world!
```

4.6 \$\$BROKER^XWBLIB

Use the \$\$BROKER^XWBLIB function in the M code called by an RPC to determine if the Broker is executing the current process. It returns:

- **1**—If this is **True**.
- **0**—If **False**.

4.7 \$\$RTRNFMT^XWBLIB

Use the \$\$RTRNFMT^XWBLIB function in the M code called by an RPC to change the return value type that the RPC returns on-the-fly. This allows you to change the return value type to any valid return value type (**Single Value**, **Array**, **Word-Processing**, **Global Array**, or **Global Instance**). It also lets you set WORD WRAP ON (#.08) field to **True** or **False**, on-the-fly, for the RPC.



REF: For more information about \$\$RTRNFMT^XWBLIB, see the *RPC Broker Developer's Guide*.

5 Broker Security Enhancement (BSE)

5.1 Introduction

This section describes the mechanism by which the Broker Security Enhancement (BSE) enables RPC Broker Delphi-based applications to make remote user/visitor connections in a more secure manner. This BSE-based mechanism subsequently replaces the current Compensation And Pension Records Interchange (CAPRI)-based mechanism for remote user/visitor access by RPC Broker Delphi-based client/server applications.

The Veterans Health Administration (VHA) information systems management and user community has expressed a need to secure access to patient information at remote sites.

Some VistA application users require access to data located at remote sites at which the users:

- Do *not* have assigned Access and Verify codes.
- Have *not* been entered into the NEW PERSON (#200) file by system administrators.
- Want to avoid having multiple Access/Verify code pairs.

The Compensation And Pension Records Interchange (CAPRI) application was the first application with these requirements. This application is used by Veterans Benefits Administration (VBA) staff to remotely access VistA data related to claims for veterans treated at any VistA site.

The CAPRI application was the first application to use the modified version of the VistA Remote Procedure Call (RPC) Broker software, which was based on the Remote Data Views (RDV) access method, as a means for obtaining such access. This access enters the user's information into the NEW PERSON (#200) file as a visitor but does *not* require an Access or Verify code for the user at the remote site. As a result of the CAPRI application, there has been an increase in the number of other applications that also require or are requesting this type of remote data access.

The goal of the Broker Security Enhancement (BSE) Project is to accomplish the following:

- Enable RPC Broker Delphi-based applications to access Remote VistA M Servers with increased security.
- Enhance the RPC Broker method used to connect to Remote VistA M Servers.
- Ensure correct information for user access to prevent the mistaken identification of an incorrect or non-existent user (spoofing) via unauthorized applications.
- Provide the ability for RPC Broker Delphi-based applications that have implemented BSE to specify their own context option.
- Allow the VistA Imaging Display Client to pull in images from remote sites without requiring credentials on the Remote VistA M Servers.

5.1.1 Features

The Broker Security Enhancement (BSE) Project provides the following features and functionality:

- Adds a step to the RPC Broker signon process to authenticate the connecting application. This also involves passing a secret encoded phrase that is established on the VistA M Server via a patch and KIDS build.
- Adds a step to the RPC Broker signon process on the Remote VistA M Server to authenticate the user by connecting back to the Authenticating VistA M Server.
- Provides the capability for remote applications to specify their own context option.

5.1.2 Architectural Scope

The architectural scope of BSE is as follows:

- **Use of Kernel Authentication**—Kernel is used as the authenticator. Kernel is a valid means of authenticating on a backend VistA M Server.
- **Client/Server-based Application Support**—This document only discusses the BSE functionality provided with VistA RPC Broker Delphi-based client/server applications.

5.2 Process Overview

The overall process to make a remote connection via an RPC Broker Delphi-based client/server application that has implemented the Broker Security enhancement (BSE) is as follows:

1. The user starts the BSE-enabled application.
2. The BSE-enabled application connects to the Authenticating VistA M Server and presents the VistA login GUI dialogue to the user.



NOTE: The Authenticating VistA M Server is identified in the CALLBACKSERVER (#.03) field in the CALLBACKTYPE (#1) Multiple field in the REMOTE APPLICATION (#8994.5) file.

3. The user enters their Kernel Access and Verify codes, is authenticated via Kernel, and is signed onto the BSE-enabled application's Authenticating VistA M Server.
4. The BSE-enabled application gets a Kernel Authentication Token for the authenticated user from the Authenticating VistA M Server. This token is eventually used by the Remote VistA M Server to obtain the necessary user information for populating a user as a "visitor" entry in the remote site's NEW PERSON (#200) file. This ensures the following:
 - The user is *not* spoofed.
 - The data at the remote site is valid.

A sample Kernel Authentication Token follows:

XWBHDL977-124367_0

5. The BSE-enabled application completes any other processing necessary to identify the Remote VistA M Server and gathers any other required information.
6. The BSE-enabled application disconnects from the Authenticating VistA M Server.
7. The BSE-enabled application performs the following tasks:
 - a. Creates a Security Pass Phrase value that is composed of the following two pieces of data:

- **Security Phrase**—A one-way hashed value that is stored in the REMOTE APPLICATION (#8994.5) file and used to identify the BSE-enabled application's file entry.



REF: For more information on the Security Phrase, see the "[Security Phrase](#)" section.

- **Kernel Authentication Token**

- b. Sets the **SecurityPhrase** property of the **RPCBroker** login component to the Security Pass Phrase value (see [Step 7a](#)), which is later used by the Remote VistA M Server to call back the Authenticating VistA M Server.
 - c. Sets the other appropriate **RPCBroker** login component properties in order to call the Remote VistA M Server.



REF: For more information on the specific **RPCBroker** login component property settings, see the "Step-By-Step Procedures to Implement BSE" section in the *RPC Broker Developer's Guide*.

8. The BSE-enabled application connects to the Remote VistA M Server with the **RPCBroker** login component passing in the encoded value of the **SecurityPhrase** property (see [Step 7](#)).



CAUTION: Remote access is only permitted at sites that have installed the application's information (including the hashed Security Phrase) into the REMOTE APPLICATION (#8994.5) file, ensuring that a rogue application *cannot* obtain access.



REF: For more information on the application's Security Phrase, see the "[Security Phrase](#)" section.

9. The Kernel software on the Remote VistA M Server performs the following tasks:
 - a. Identifies and hashes the decoded value of the **RPCBroker** login component's **SecurityPhrase** property (see Steps [7a](#) and [7b](#)).
 - b. Uses the hashed value of the BSE-enabled application's Security Pass Phrase to identify the application's entry in the REMOTE APPLICATION (#8994.5) file.



NOTE: Included in that entry is the mechanisms for contacting the Authenticating VistA M Server.

- c. Connects to the Authenticating VistA M Server passing in the Kernel Authentication Token that identifies the user.
 - d. Obtains the user demographic information from the Authenticating VistA M Server. This user demographic information is used to establish the user as a remote user/visitor.
 - e. Disconnects from the Authenticating VistA M Server.
 - f. Uses the demographic information obtained from the Authenticating VistA M Server to set up the user as a visitor entry on the Remote VistA M Server. It creates or matches an entry in the NEW PERSON (#200) file and provides the visitor with the context option specified for the BSE-enabled application in the REMOTE APPLICATION (#8994.5) file.
10. The BSE-enabled application is notified by the **RPCBroker** login component that it successfully connected, and that the user is signed on to the Remote VistA M Server. The user can then continue with any processing necessary on the Remote VistA M Server. If for some reason the user signon fails on the Remote VistA M Server, the user is prompted to enter a valid Access and Verify code on the Remote VistA M Server. If the user cancels the signon, he/she is prompted with a signon cancellation dialogue box.



REF: For more information on the REMOTE APPLICATION (#8994.5) file, see the "[REMOTE APPLICATION \(#8994.5\) File](#)" section.

If any of the following error conditions exist, the user is prompted with a regular GUI signon dialogue instructing them to enter their Access and Verify codes:


- No entry for the application in the REMOTE APPLICATION (#8994.5) file.
- No match for the Kernel Authentication Token.
- Cannot connect to the Authenticating VistA M Server.

The Remote VistA M Server connects to the Authenticating VistA M Server and passes in the Kernel Authentication Token identifying the user. The Authenticating VistA M Server responds back by returning the demographic information necessary to establish the user as a remote user. The Remote VistA M Server disconnects from the Authenticating VistA M Server and sets up the user's profile as a visitor entry, including the necessary context option specified for the application in the REMOTE APPLICATION (#8994.5) file.

The BSE-enabled application is notified that the user is signed on and continues processing as normal.

[Table 8](#) lists the two classes of applications that use this BSE authentication mechanism:

Table 8: BSE—Application Authentication Server Class Types

Application Class	Description
Single Server Authentication	<p>Applications that require users to authenticate against a single VistA M Server and determine the remote locations to be accessed (e.g., CAPRI).</p> <p>For those applications where the users all authenticate on a single VistA M Server, the application only needs to specify the Uniform Resource Locator (URL) for its VistA M Server and one or more methods for connecting to it (including port number[s]) in the CALLBACKTYPE Multiple of the REMOTE APPLICATION (#8994.5) file.</p>
Multiple Server Authentication	<p>Applications that require users to authenticate at their local medical center or other site (e.g., VistAWeb or other Web-based applications).</p> <p>For those applications where each user authenticates on multiple/different VistA M Servers, the application needs to obtain both a Kernel Authentication Token and the demographic data necessary for identifying or adding a remote user/visitor during the authentication process on the Authenticating VistA M Server. The application passes in the Kernel Authentication Token and application Security Pass Phrase, as described above (see the "Process Overview" topic). The REMOTE APPLICATION (#8994.5) file contains an address for the Web-based application and the Remote VistA M Server returns the Kernel Authentication Token to the application and expects it to return the demographic information associated with that Kernel Authentication Token. This requires the application to keep the Kernel Authentication Token and demographic data in a location that is accessible by the application until the demographic data has been provided to the Remote VistA M Server.</p> <div>  <p>RECOMMENDATION: VistA Infrastructure (VI) highly encourages that other <i>non-Web-based</i> applications use a single server rather than multiple servers for user authentication.</p> </div>

5.2.1 Process Diagrams

[Figure 14](#) illustrates the BSE process sequence flow:

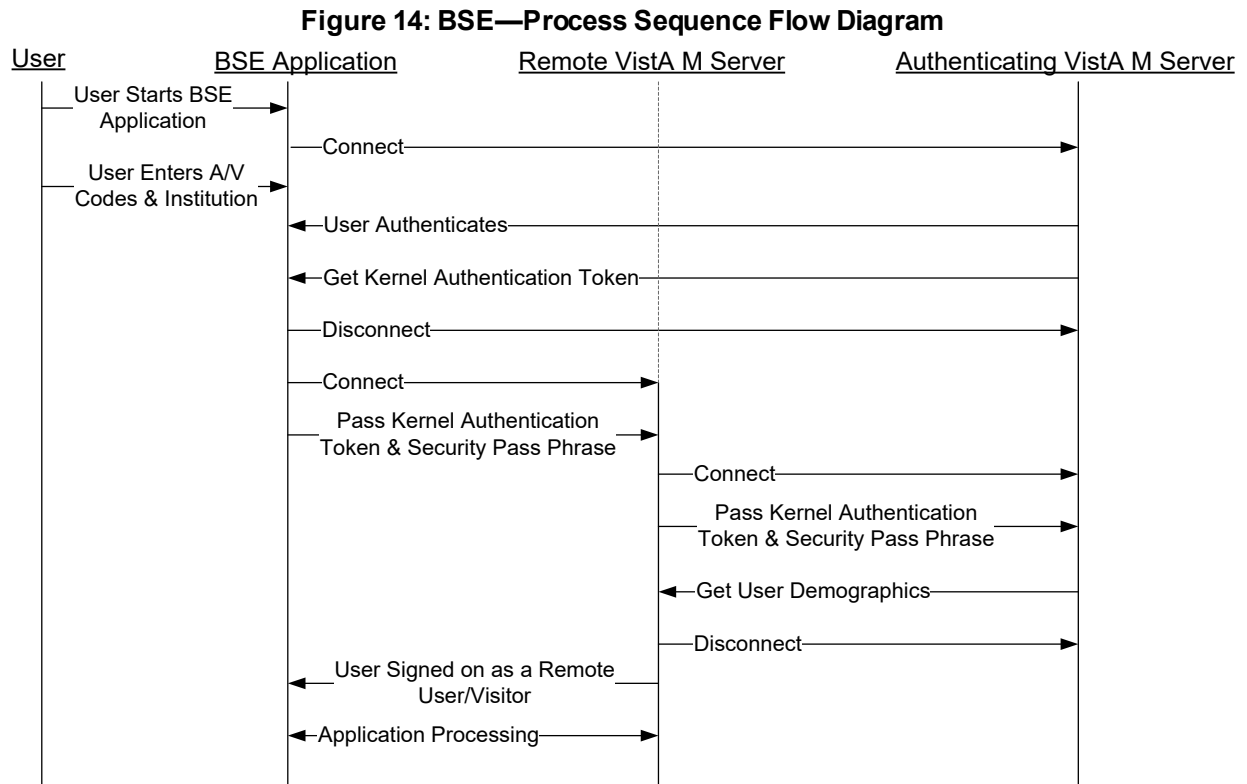
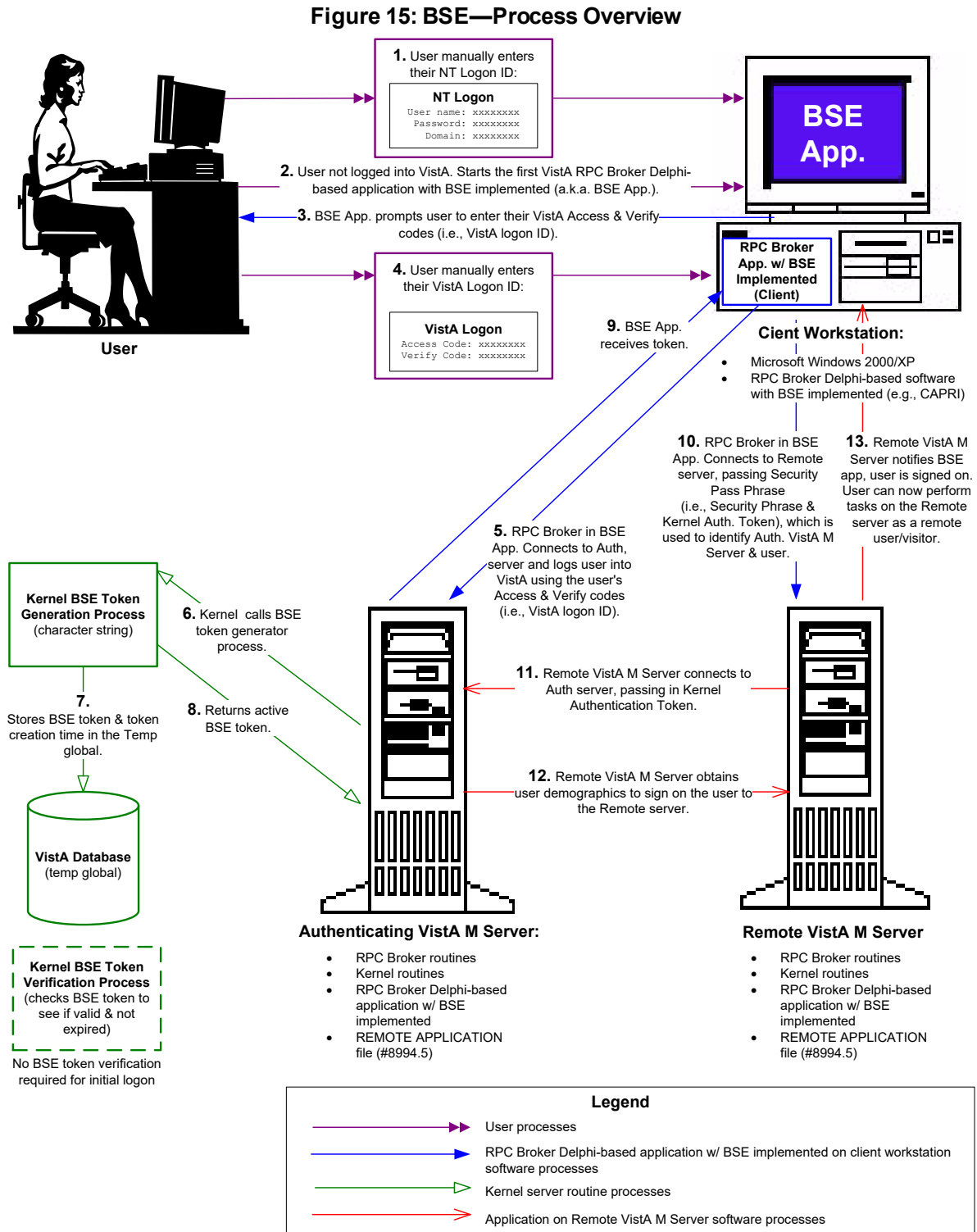


Figure 15 illustrates the BSE process overview:



5.3 BSE-related VistA Applications and Modules

This section describes the new or modified functionality made to the BSE-related software applications and modules as listed in [Table 9](#).


An RPC Broker Delphi-based and BSE-enabled VistA application comprises software that has been re-compiled using the **RPCBroker** login component, modified for BSE. BSE capability comes into play when you are using a BSE-enabled application (e.g., Compensation And Pension Records Interchange [CAPRI] or VistAWeb).



REF: For information on how to implement BSE in VistA RPC Broker Delphi-based client/server applications, see the "Implementing BSE in VistA RPC Broker-based Applications," in the *RPC Broker Developer's Guide*.

This section discusses in more detail the various software applications and modules that, together, provide for BSE functionality:

Table 9: BSE—Software Applications and Modules

Application/Module	Location	Description
VistA M Server	VistA M Server	<p>This is the "backend server" where the Kernel and RPC Broker software act as the authentication source for all VistA applications (i.e., client/server, rich client, Web, and roll-and-scroll applications). The VistA M Server also executes remote procedure calls (RPCs) and provides other functions to VistA applications.</p> <p> REF: For a list of BSE-related Vista M Server patches, see the "BSE Installation Instructions for Developers" section in the <i>RPC Broker Developer's Guide</i>.</p>
Client/Server Login Component: RPC Broker	Client (Developer workstations only)	<p>The RPCBroker login component allows client/server applications to authenticate against the VistA M Server and obtain a persistent connection over which remote procedure calls (RPCs) are executed. This component is modified in BSE to be more secure when accessing data at remote sites.</p> <p>RPC Broker-based applications using remote or visitor access (e.g., Compensation And Pension Records Interchange [CAPRI], VistAWeb) <i>must</i> invoke this modified RPCBroker login component to implement the Broker Security Enhancement (BSE).</p>



REF: For the specific software patches required for the implementation of BSE, see the “BSE Installation Instructions for Developers” section in the *RPC Broker Developer’s Guide*.

5.4 Kernel—Authentication Interface to VistA

Authentication is the process of verifying a user identity to ensure that the person requesting access to a VistA system (e.g., clinical information system) is, in fact, that person to whom entry is authorized.

Currently, Kernel on the VistA M Server is the approved method to provide both Authentication and Authorization (AA) services for all VistA applications. Kernel was assessed as the most straightforward and timely approach to also be used for remote signon authentication in BSE. By using Kernel as the authenticator for BSE, the NEW PERSON (#200) file continues to serve as the single user data store for VistA and BSE.

Some potential advantages to employing Kernel as the AA source include the following:

- Ease of file maintenance by system administrators.
- Provides a single point of user management for existing and new VistA RPC Broker Delphi-based applications.
- Allows the use of an existing credential (i.e., the Access and Verify code) for Authentication and Authorization, rather than introducing a new security credential.
- Ease of coding requirements by application developers.
- Avoids an additional user store, which simplifies the migration to any future AA solutions.

The BSE functionality for Kernel was introduced with Kernel Patch XU*8.0*404 (server-side). The BSE functionality includes the creation of a Kernel Authentication Token. The Kernel Authentication Token is generated once a user has been initially authenticated on the Authenticating VistA M Server via their Access and Verify codes. This Kernel Authentication Token can then be used to authenticate a user on a Remote VistA M Server.

5.5 RPC Broker

The RPC Broker software consists of both a client and server software piece.

5.5.1 Client

The **RPCBroker** login component is embedded in a Embarcadero Delphi-based rich client/server application (e.g., Compensation And Pension Records Interchange [CAPRI]). The **RPCBroker** login component is used to connect the application running on a Microsoft Windows client workstation to the VistA M Server. This connection allows data retrieval from the VistA M Server database. The **RPCBroker** login component uses Kernel's Access and Verify codes to authenticate a user to VistA.

The BSE functionality for the **RPCBroker** login component was introduced with RPC Broker Patch XWB*1.1*45 (client-side) and Kernel Patch XU*8.0*404 (server-side). BSE functionality includes the addition of a property to the **RPCBroker** login component that allows applications to pass an application's Security Phrase and Kernel Authentication Token, which is referred to in this documentation as the Security Pass Phrase. Thus, when a VistA RPC Broker Delphi-based application, such as CAPRI, is recompiled with the BSE-updated **RPCBroker** login component and other required code modifications are made, that application would then become capable of accessing Remote VistA M Servers without requiring users to re-enter their Access and Verify codes.

5.5.2 Server

In order to implement BSE and use the RPC-Broker callback type, the central Authenticating VistA M server *must* run the RPC Broker as a TCPIP service. The *Non*-callback RPC Broker Listener/TCPIP service is distributed and described with RPC Broker Patch XWB*1.1*35 and was updated with XWB*1.1*44.



REF: For more information on the RPC Broker and TCPIP service setup, see the RPC Broker Patches XWB*1.1*35 and 44 on FORUM and the RPC Broker documentation, specifically the *RPC Broker TCP/IP Supplement*, located on the VDL at the following Web address: <http://www.va.gov/vdl/application.asp?appid=23>




REF: For more detailed information on the application developer procedures and code modifications needed to implement BSE in RPC Broker Delphi-based applications, see the "Implementing BSE in VistA RPC Broker-based" section in the *RPC Broker Developer's Guide*.

5.6 REMOTE APPLICATION (#8994.5) File

The REMOTE APPLICATION (#8994.5) file was released with RPC Broker Patch XWB*1.1*45. This file helps better secure remote user/visitor access to Remote Vista M Servers initiated by RPC Broker Delphi-based GUI applications. Remote user/visitor access permits applications where users need to access a large number of sites and do so *without* requiring separate Access and Verify codes at each target remote site.

The REMOTE APPLICATION (#8994.5) file contains the fields listed in [Table 10](#):

Table 10: Fields in the REMOTE APPLICATION (#8994.5) File

Field Name	Field Number	Description
NAME	.01	(required) This is the name for the RPC Broker Delphi-based application that requires remote user/visitor access. The name <i>must</i> be from 3 to 30 characters, not numeric or starting with punctuation.
CONTEXTOPTION	.02	(required) This is the name of the context option (i.e., client/server or "B"-type option) that the application users need. The name <i>must</i> be from 3 to 45 characters. The user is signed on as a visitor and given this context option as a secondary menu option.
APPLICATIONCODE	.03	(required) This is the hashed value for an application's Security Phrase.  REF: For more information on the Security Phrase, see the " Security Phrase " section.
CALLBACKTYPE	1	(required) This is a Multiple field. It can contain multiple values describing the mechanisms by which the Remote Vista M Server can contact the application's Authenticating Vista M Server to obtain the demographic information. It consists of the subfields described below.
CALLBACKTYPE (CALLBACKTYPE Multiple)	.01	(required) This field indicates the mechanisms by which the server should contact the Authenticating Vista M Server to obtain information necessary to sign the current user onto the current server. The values for this field are: <ul style="list-style-type: none"> • R—RPC Broker TCP/IP connection • M—M-to-M Broker connection • H—HyperText Transport Protocol (HTTP) connection • S—Station-number callback
CALLBACKPORT (CALLBACKTYPE Multiple)	.02	(required) This is the port number (3 - 5 characters) to be used for the callback connection to the Authenticating Vista M Server for the CALLBACKTYPE (#.01) specified.
CALLBACKSERVER	.03	(required) This is the server designation (address) to be

Field Name	Field Number	Description
(CALLBACKTYPE Multiple)		used for the callback to the Authenticating VistA M Server for the CALLBACKTYPE (#.01) specified. This should be a Domain Name Service (DNS) name-based address rather than an Internet Protocol (IP) address, because IP addresses can change. It should be a server name ending in <REDACTED>.VA.GOV or <REDACTED>.VA.GOV . The DNS servers resolve the name, and thus, ensure that the site is a valid VistA M Server.
URLSTRING (CALLBACKTYPE Multiple)	.04	<p>(optional) This field holds the text that should follow the SERVER ADDRESS (#.03) field for HTTP connections to obtain the information for the Kernel Authentication Token passed in for a REMOTE APPLICATION connection. If the complete Uniform Resource Locator (URL) to be used for the callback is:</p> <pre>http://<REDACTED>.va.gov/some/kind/of/location/somePage.aspx</pre> <p>The CALLBACKSERVER (#.03) field could be:</p> <pre><REDACTED>.va.gov</pre> <p>and the URLSTRING would be:</p> <pre>some/kind/of/location/somePage.aspx</pre> <p>This field is only used if the CALLBACKTYPE filed (#.01) value is H for HTTP.</p>



REF: For more information on the REMOTE APPLICATION (#8994.5) file, see the "Files" section in the *RPC Broker Technical Manual*.

5.7 Security Phrase

The Security Phrase is an RPC Broker Delphi-based application's entry into the REMOTE APPLICATION (#8994.5) file. The Security Phrase is a general phrase that is known only to the application that created it. When it is stored in the REMOTE APPLICATION (#8994.5) file, it *must* be hashed. This one-way hashed value, which is the result of a call to the \$\$EN^XUSHSH(phrase) API, is entered into the APPLICATIONCODE (#.03) field in the REMOTE APPLICATION (#8994.5) file for the application.

This Security Phrase is combined with the Kernel Authentication Token to make up the Security Pass Phrase, which is then stored in the **SecurityPhrase** property of the **RPCBroker** login component.



CAUTION: It is important to realize that the Security Phrase identifies only those applications that are authorized to perform remote user/visitor access. Thus, the stored value of the Security Phrase is a one-way hash so that other rogue applications *cannot* mimic an application and access the Remote VistA M Server.



RECOMMENDATION: Since the Security Phrase is the application's identifier, VistA Infrastructure (VI) *recommends* developers identify the Security Phrase as a const value in an include file in any RPC Broker Delphi-based program implementing BSE. A substitute include file containing a phrase similar to the Security Phrase should then be included with release of the source code.

5.8 Kernel Authentication Token

The Kernel Authentication Token is generated by the same code used to generate handles (i.e., a unique text string that is used to identify a specific user for which it was generated) for other purposes used in the RPC Broker software. Once created, the token is stored in the ^XTMP temporary global. The basic format of the token (handle) is as follows:

`XWBHDLnnnn-nnnnnn_n`

The "XWBHDL" indicates that it is an RPC Broker handle; where "XWB" is the RPC Broker namespace and "HDL" indicates that it is a handle.

The following is an example of a Kernel Authentication Token:

`XWBHDL977-124367_0`

6 Debugging and Troubleshooting

6.1 How to Debug Your Client Application

Beside the normal debugging facilities provided by Delphi, you can also invoke a debug mode, so that you can step through your code on the client side and your RPC code on the M server side simultaneously.

To invoke the debug mode, perform the following procedure:

1. On the client side, set the **DebugMode** property on the **TRPCBroker** component to **True**.
2. Switch over to the VistA M Server and set any break points in the routines being called in order to help isolate the problem.
3. Issue the M debug command (e.g., **ZDEBUG**) or follow instructions in the InterSystems Caché documentation on “Debugging with the Caché Debugger.”
4. Start the following VistA M Server process:

```
>D DEBUG^XWBTCPM
```

5. Enter a unique Listener port number (i.e., a port number *not* in general use).
6. Switch over to the client application and connect the client application to the VistA M Server using the server’s IP address and the port number you entered [Step 5](#).
7. You can now step through the code on your client and simultaneously step through the code on the VistA M Server side for any RPCs that your client calls.

6.1.1 RPC Error Trapping

M errors on the VistA M Server that occur during RPC execution are trapped by the use of M and Kernel error handling. In addition, the M error message is sent back to the Delphi client. Delphi raises an exception **EBrokerError** and a popup box displaying the error. At this point, RPC execution terminates, and the channel is closed.

6.2 Troubleshooting Connections

6.2.1 Identifying the Listener Process on the Server

On InterSystems Caché systems, where the Broker Listener is running and the **System Status** [XUSTATUS] menu option is available, the Listener process name is:

```
[TCP]####
```

Where **####** is the port number being listened to. This should help quickly locate Listener processes when troubleshooting any connection problems.

On systems with greater security or with listener processes started by Linux **xinetd.d** scripts, the following commands can be helpful:

- List all **xinetd.d** scripts:
`>!ls -al /etc/xinetd.d`
- List **xinetd.d** scripts containing string “vis”:
`>!ls -al /etc/xinetd.d | grep vis`
- Display script “vis_rpc”:
`>!cat /etc/xinetd.d/vis_rpc`
- Look for listener running on port <REDACTED>:
`>!netstat -an | grep :<REDACTED>`

6.2.2 Identifying the Handler Process on the Server

On InterSystems Caché systems the name of a Handler process for IPv4 is:

```
[TCP|nnn.nnn.nnn.nnn: #####
```

Where *nnn.nnn.nnn.nnn* is the client IPv4 address and ##### is the port number.

Alternatively, for IPv6:

```
[TCP|hhhh:hhhh::hhhh:#####
```

Where *hhhh* represents the hexadecimal segments of the client IPv6 address and ##### is the port number.

6.2.3 Testing Your RPC Broker Connection

To test the RPC Broker connection from your workstation to the M Server, use the RPC Broker Diagnostic Program (**rpctest.exe**).



REF: For a complete description of the RPC Broker Diagnostic program, see the “Troubleshooting” chapter in the *RPC Broker Systems Management Guide*.

7 RPC Broker and Delphi

The following sections highlight changes made to or comments about the RPC Broker to accommodate a particular version of Delphi.



RECOMMENDATION: To avoid problems with the BDK, it is *recommended* for all Delphi packages that you accept the default directory after compiling the Broker Development Kit (BDK) on a workstation.

7.1 Delphi 10.4, 10.3, 10.2, 10.1, 10.0, and XE8 Packages

7.1.1 Delphi Starter Edition—*Not* Recommended for BDK Development

Delphi 10.4, 10.3, 10.2, 10.1, 10.0, and XE8 comes in three flavors:

- Starter
- Professional
- Enterprise



RECOMMENDATION: It is *recommended* that you use either the Professional or Enterprise version of Delphi to develop applications using the RPC Broker.

This version of the BDK requires the Professional or Enterprise Edition. The Starter editions are targeted mainly at students, and as such, leave out many features. We do *not* recommend using any of the Starter editions of Delphi for RPC Broker development at this time. Delphi Starter Edition does *not* ship the following:

- OpenHelp help system—Allow easy integration of 3rd party component help with Delphi's own internal component help.
- VCL source code unit (i.e., **dsgnintf.pas** file)—**RPCBroker** component has a dependency on a VCL source code unit. Delphi Starter Editions do *not* ship VCL source code unit in either **.PAS** or **.DCU** form; however, VCL Source code units are available in Delphi Professional and Enterprise editions.



NOTE: When installing Delphi Professional or Enterprise editions, make sure you leave the VCL Source installation option selected.

7.1.2 XWB_RXE#.bpl File

This **RunTime** package contains the source code for the standard **RPCBroker** components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation. Shown are the default paths for various versions of Delphi, where # represents the version number. If you have changed any default paths, your files may be in a different location:

- C:\Users\Public\Public Documents\Embarcadero\Studio\16.0\Bpl\XWB_RXE8.bpl
- C:\Users\Public\Public Documents\Embarcadero\Studio\17.0\Bpl\XWB_RunTime.bpl
- C:\Users\Public\Public Documents\Embarcadero\Studio\18.0\Bpl\XWB_RunTime.bpl

7.1.3 XWB_DXE#.bpl File

This **DesignTime** package contains the installed components for the standard **RPCBroker** and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation. Shown are the default paths for various versions of Delphi, where # represents the version number. If you have changed any default paths, your files may be in a different location:

- C:\Users\Public\Public Documents\Embarcadero\Studio\16.0\Bpl\XWB_DXE8.bpl
- C:\Users\Public\Public Documents\Embarcadero\Studio\17.0\Bpl\XWB_DesignTime.bpl
- C:\Users\Public\Public Documents\Embarcadero\Studio\18.0\Bpl\XWB_DesignTime.bpl

8 RPC Broker Dynamic Link Library (DLL)

8.1 DLL Interface

The RPC Broker provides a Dynamic Link Library (DLL) interface, which acts like a “shell” around the Delphi **TRPCBroker** component. The DLL is contained in the **BAPI32.DLL** file.

The DLL interface enables client applications, written in any language that supports access to Microsoft Windows DLL functions, to take advantage of all features of the **TRPCBroker** component. This allows programming environments other than Embarcadero Delphi to make use of the **TRPCBroker** component. All of the communication to the server is handled by the **TRPCBroker** component, accessed via the DLL interface.

The DLL interface has *not* been updated to support Secure Shell (SSH) or IPv4/IPv6 dual-stack environments.

8.1.1 Exported Functions

The complete list of functions exported in the DLL is provided in the BDK *RPC Broker Developer's Guide*. Functions are provided in the DLL for:

- Creating and destroying RPC Broker components.
- Setting and retrieving RPC Broker component properties.
- Executing RPC Broker component methods.

8.1.2 Header Files Provided

[Table 11](#) lists the header files that provide correct declarations for DLL functions:

Table 11: Header Files that Provide Correct Declarations for DLL Functions

Language	Header File
C	BAPI32.H
C++	BAPI32.HPP
Visual Basic	BAPI32.BAS

8.1.3 Return Values from RPCs

Results from an RPC executed on an M server are returned as a text stream. This text stream may or may *not* have embedded <CR><LF> character combinations.

When you call an RPC using the **TRPCBroker** component for Delphi, the text stream returned from an RPC is automatically parsed and returned in the **TRPCBroker** component's **Results** property as follows:

Table 12: TRPCBroker Component's Results Property

Results Stream Contains <CR><LF> Combinations	Results Location/Format (assumes RPC's WORD WRAP ON field is True if RPC is Global Array or Word-processing type)
Yes	Results nodes; split based on <CR><LF> delimiter
No	Results[0]

When you call an RPC using the DLL interface, the return value is the unprocessed text stream, which may or may *not* contain <CR><LF> combinations. It is up to you to parse out what would have been individual Results nodes in Delphi, based on the presence of any <CR><LF> character combinations in the text stream.

8.1.4 COTS Development and the DLL

The Broker DLL serves as the gateway to the REMOTE PROCEDURE (#8994) file for *non*-Delphi client/server applications. In order to use any RPCs *not* written specifically by the client application (e.g., CONSULTS FOR A PATIENT, USER SIGN-ON RPCs, or the more generic VA FileMan RPCs), you *must* call the RPC Broker DLL with input parameters defined and results accepted in the formats required by the RPC being called.

Therefore, to use the Broker DLL interface you *must* determine the following information for each RPC you plan to use:

- How does the RPC expect input parameters, if any, to be passed to it?
- Will you be able to create any input arrays expected by the RPC in the same format expected by the RPC?
- What does the results data stream returned by the RPC look like?

Glossary

Table 13: Glossary of Terms and Acronyms

Term	Definition
BDK	Broker Development Kit.
BSE	Broker Security Enhancement.
Client	A single term used interchangeably to refer to the user, the workstation, and the portion of the program that runs on the workstation. In an object-oriented environment, a client is a member of a group that uses the services of an unrelated group. If the client is on a local area network (LAN), it can share resources with another computer (server).
Component	An object-oriented term used to describe the building blocks of GUI applications. A software object that contains data and code. A component may or may <i>not</i> be visible. These components interact with other components on a form to create the GUI user application interface.
DHCP	D ynamic H ost C onfiguration P rotocol.
DLL	<p>Dynamic Link Library. A DLL allows executable routines to be stored separately as files with a DLL extension. These routines are only loaded when a program calls for them. DLLs provide several advantages:</p> <ul style="list-style-type: none"> • Help save on computer memory, since memory is only consumed when a DLL is loaded. They also save disk space. With static libraries, your application absorbs all the library code into your application, so the size of your application is greater. Other applications using the same library also carry this code around. With the DLL, you do <i>not</i> carry the code itself; you have a pointer to the common library. All applications using it will then share one image. • Ease maintenance tasks. Because the DLL is a separate file, any modifications made to the DLL do not affect the operation of the calling program or any other DLL. • Help avoid redundant routines. They provide generic functions that can be used by a variety of programs.
GUI	G raphical U ser I nterface. A type of display format that enables users to choose commands, initiate programs, and other options by selecting pictorial representations (icons) via a mouse or a keyboard.
IAM	Identity and Access Management.

Term	Definition
Icon	A picture or symbol that graphically represents an object or a concept.
PIN	Personal Identification Number.
PKI	Public Key Encryption.
Remote Procedure Call	A remote procedure call (RPC) is essentially M code that can take optional parameters to do some work and then return either a single value or an array back to the client application.
SAML	Security Assertion Markup Language. An XML-based industry standard for communicating identities over the Internet.
Server	The computer where the data and the Business Rules reside. It makes resources available to client workstations on the network. In VistA, it is an entry in the OPTION (#19) file. An automated mail protocol that is activated by sending a message to a server at another location with the “ S.server ” syntax. A server’s activity is specified in the OPTION (#19) file and can be the running of a routine or the placement of data into a file.
SSH	Secure Shell.
SSO/UC	Sign-On/User Context.
STS	Secure Token Service.
User Access	This term is used to refer to a limited level of access to a computer system that is sufficient for using/operating software, but does not allow programming, modification to data dictionaries, or other operations that require programmer access. Any of VistA’s options can be locked with a security key (e.g., XUPROGMODE, which means that invoking that option requires programmer access). The user’s access level determines the degree of computer use and the types of computer programs available. The Systems Manager assigns the user an access level.
User Interface	The way the software is presented to the user, such as Graphical User Interfaces that display option prompts, help messages, and menu choices. A standard user interface can be achieved by using Borland’s Delphi Graphical User Interface to display the various menu option choices, commands, etc.
VistA	Veterans Health Information Systems and Technology Architecture.
Window	An object on the screen (dialogue) that presents information such as a document or message.
XML	eXtensible Markup Language.



REF: For a list of commonly used terms and definitions, see the OIT Master Glossary VA Intranet Website.

For a list of commonly used acronyms, see the VA Acronym Lookup Intranet Website.

Index

\$

\$\$BROKER^XWBLIB, 26
\$\$EN^XUSHSH API, 39
\$\$RTRNFMT^XWBLIB, 27

^

^XTMP Global, 40

A

About this Version of the BDK, 1
Acronyms
 Intranet Website, 49
APIs
 \$\$BROKER^XWBLIB, 26
 \$\$EN^XUSHSH, 39
 \$\$RTRNFMT^XWBLIB, 27
APP PROXY ALLOWED (#.11) Field, 16
Application.Run Method, 24
APPLICATIONCODE (#.03) Field, 38, 39
Architectural Scope, 29
Assumptions, xix
Authentication
 Interface to VistA
 Kernel, 36
 Kernel Authentication Token, 29, 31, 32,
 36, 37, 39, 40
 Sample, 30

B

Backward Compatibility Issues, 3
BAPI32.BAS File, 45
BAPI32.DLL File, 45
BAPI32.H File, 45
BAPI32.HPP File, 45
Broker
 Component, 30, 31, 35, 36, 37, 39
 Patches
 XWB*1.1*45, 37
BrokerExample, 20
BROKEREXAMPLE.EXE, 20

BSE

 Introduction, 28
 Project Overview, 28
 Scope, 29
 VistA Applications/Modules, 35
Bypassing Security for Development, 19

C

C Language, 45
C++ Language, 45
Call Method, 6, 18
CALLBACKPORT (#.02) Field
 CALLBACKTYPE (#1) Multiple Field,
 38
CALLBACKSERVER (#.03) Field
 CALLBACKTYPE (#1) Multiple Field,
 29, 38
CALLBACKTYPE (#.01) Field
 CALLBACKTYPE (#1) Multiple Field,
 38
CALLBACKTYPE (#1) Multiple Field, 29,
 38
 CALLBACKPORT (#.02) Field, 38
 CALLBACKSERVER (#.03) Field, 29,
 38
 CALLBACKTYPE (#.01) Field, 38
 URLSTRING (#.04) Field, 39
Callout Boxes, xvi
Calls
 Discrete, 17
 Silent, 17
CAPRI, 28, 35, 37
ClearParameters Property, 5
ClearResults Property, 5
Commonly Used Terms, xvii
Compatibility Issues, 3
Components
 RPC Broker, 30, 31, 35, 36, 37, 39
 RPC Broker Components for Delphi, 4
 RPCBroker, 35
 TCCOWRPCBroker, 9
 TRPCBroker, 4
 TXWBRichEdit, 9

- TXWSSOiToken, 10
- Connect To, 22
- Connected Property, 5
- Connection
 - Testing Your RPC Broker Connection, 42
- Contents, x
- CONTEXTOPTION (#.02) Field, 38
- COTS Development and the DLL, 46
- Create Your Own RPCs
 - Preliminary Considerations, 11
 - Process, 12
- CreateContext Method, 6, 19, 25

D

- Data Dictionary
 - Data Dictionary Utilities Menu, xviii
 - Listings, xviii
- DEBUG^XWBTCMP, 41
- Debugging, 41
 - Error Trapping, 41
 - How to Debug Your Client Application, 41
 - Identifying
 - Handler Process on the Server, 42
 - Listener Process on the Server, 41
 - Testing Your RPC Broker Connection, 42
- DebugMode Property, 41
- DECRYPT^XUSRB1, 26
- Decrypt Method, 26
- Decryption Functions, 26
- Delphi, 43
 - Starter Edition, 43
- Delphi Components
 - RPC Broker, 4
- Demographics, 31, 32, 38
- DI DDU Menu, xviii
- Diagnostic Program, 42
- DILIST Option, xviii
- Disclaimers, xv
 - Software, xiv
- Discrete Calls, 17
- DLL
 - COTS Development and the DLL, 46
 - Exported Functions, 45
 - Header Files, 45
 - Interface, 45

- Documentation
 - Revisions, ii
 - Symbols, xv
- Documentation Conventions, xv
- Documentation Navigation, xvii
- Dynamic Link Library (DLL), 45

E

- EBrokerError, 41
- ENCRYPT^XUSRB1, 26
- Encrypt Method, 26
- Encryption Functions, 26
- Entry in the Remote Procedure File, 16
- Error Message Handling, 41
- Execute an RPC from a Client Application,
 - How to, 17
- Exported
 - DLL Functions, 45

F

- Features, 29
- Fields
 - APP PROXY ALLOWED (#.11), 16
 - APPLICATIONCODE (#.03), 38, 39
 - CALLBACKPORT (#.02)
 - CALLBACKTYPE (#1) Multiple Field, 38
 - CALLBACKSERVER (#.03)
 - CALLBACKTYPE (#1) Multiple Field, 29, 38
 - CALLBACKTYPE (#.01)
 - CALLBACKTYPE (#1) Multiple Field, 38
 - CALLBACKTYPE (#1) Multiple, 29, 38
 - CALLBACKPORT (#.02) Field, 38
 - CALLBACKSERVER (#.03) Field, 29, 38
 - CALLBACKTYPE (#.01) Field, 38
 - URLSTRING (#.04) Field, 39
 - CONTEXTOPTION (#.02), 38
 - NAME (#.01), 16, 38
 - PARAMETER TYPE (#8994.02,.02), 17
 - RETURN VALUE TYPE (#.04), 16
 - ROUTINE (#.03), 16
 - TAG (#.02), 16

URLSTRING (#.04)

CALLBACKTYPE (#1) Multiple Field,
39

WORD WRAP ON (#.08), 13, 16, 27

Figures, xii

Files

BAPI32.BAS, 45

BAPI32.DLL, 45

BAPI32.H, 45

BAPI32.HPP, 45

Header Files, 45

NEW PERSON (#200), 14, 28, 29, 31, 36

OPTION (#19), 12, 19

REMOTE APPLICATION (#8994.5), 29,
30, 31, 32, 38, 39

REMOTE PROCEDURE (#8994), 11, 12,
16, 46

SECURITY KEY (#19.1), 19

XWB_DXE#.bpl, 44

XWB_RXE#.bpl, 44

First Input Parameter for RPCs (Required),
12

Functions

Decryption, 26

Encryption, 26

Exported with DLL, 45

Piece, 25

Translate, 25

G

GetServerInfo Method, 5, 7, 22

Globals

^XTMP, 40

Glossary, 47

Intranet Website, 49

H

HASH, 26

Header Files, 45

Help

At Prompts, xviii

Online, xviii

Question Marks, xviii

History

Revisions, ii

Home Pages

Acronyms Intranet Website, 49

Adobe Website, xix

Glossary Intranet Website, 49

RPC Broker Website, xix

VA Software Document Library (VDL)

RPC Broker Home Page Web Address,
37

VA Software Document Library (VDL)
Website, xix

How to

Connect to an M Server, 7

Debug Your Client Application, 41

Execute an RPC from a Client
Application, 17

Obtain Technical Information Online,
xviii

Register an RPC, 19

Use this Manual, xiv

I

Identifying

Handler Process on the Server, 42

Listener Process on the Server, 41

Input Parameter Types for RPCs (Optional),
15

Intended Audience, xiv

Interface

DLL, 45

Introduction, 1, 28

Issues

Backward Compatibility, 3

K

Kernel, 29, 36

Authentication

Interface to VistA, 36

Token, 29, 31, 32, 36, 37, 39, 40

Sample, 30

Patches

XU*8.0*404, 36

L

LAN, 47

- List File Attributes Option, xviii
- List PType, 15
- ListenerPort Property, 5, 22
- Literal PType, 15
- lstCall Method, 6, 18

M

- M Emulation Functions, 25
- M Entry Points for RPC Examples, 15
- Menus
 - Data Dictionary Utilities, xviii
 - DI DDU, xviii
 - System Status Menu, 41
 - XUSTATUS, 41
- Message Handling, Errors, 41
- Methods
 - Application.Run, 24
 - Call, 6, 18
 - CreateContext, 6, 19, 25
 - Decrypt, 26
 - Encrypt, 26
 - GetServerInfo, 5, 22
 - lstCall, 6, 18
 - SplashClose, 23, 24
 - SplashOpen, 23, 24
 - strCall, 6, 18
 - TRPCBroker Component, 4
- MFUNSTR.PAS, 25
- Microsoft Windows Registry, 8, 22
- Mult Property, 15, 17
- Multiple Server Authentication, 32

N

- NAME (#.01) Field, 16, 38
- NEW PERSON (#200) File, 14, 28, 29, 31, 36

O

- Obtaining
 - Data Dictionary Listings, xviii
- Online
 - Documentation, xviii
 - Technical Information, How to Obtain, xviii

- Online Code Samples (RPCs), 20
- OPTION (#19) File, 12, 19
- Options
 - Data Dictionary Utilities, xviii
 - DI DDU, xviii
 - DILIST, xviii
 - List File Attributes, xviii
 - System Status Menu, 41
 - XUSTATUS, 41
- Orientation, xiv
- Other RPC Broker APIs, 22

P

- Param Property, 5, 15, 17, 18
- PARAMETER TYPE (#8994.02,.02) Field, 17
- Parameters
 - TimeOut, 24
- Patches
 - Revisions, ix
 - XU*8.0*404, 36
 - XWB*1.1*45, 37
- Piece Function, 25
- Process
 - Diagrams, 33
 - Overview, 29
- Product Support (PS)
 - Anonymous Directories, xx
- Programs
 - BROKEREXAMPLE.EXE, 20
- Properties
 - ClearParameters, 5
 - ClearResults, 5
 - Connected, 5
 - DebugMode, 41
 - ListenerPort, 5, 22
 - Mult, 15, 17
 - Param, 5, 15, 17, 18
 - RemoteProcedure, 5, 18
 - Results, 5, 18
 - SecurityPhrase, 30, 31, 39
 - Server, 5, 22
 - SSHPort, 5
 - SSHPw, 5
 - SSHUser, 5, 22
 - SSHUseSecureConnection, 5

- TRPCBroker Component, 4
- Value, 17

PS

- Anonymous Directories, xx

PTypes

- List, 15
- Literal, 15
- Reference, 15

Q

- Question Mark Help, xviii

R

- Reference PType, 15
- Registering RPCs, 19
- Registry, 8, 22
- Relationship between an M Entry Point and an RPC, 11
- REMOTE APPLICATION (#8994.5) File, 29, 30, 31, 32, 38, 39
- Remote Data Views, 28
- REMOTE PROCEDURE (#8994) File, 11, 12, 16, 46
- Remote Procedure Calls (RPCs), 11
- RemoteProcedure Property, 5, 18
- Results Property, 5, 18
- RETURN VALUE TYPE (#.04) Field, 16
- Return Value Types for RPCs, 13
- Return Values from RPCs, 46
- Revision History, ii
 - Documentation, ii
 - Patches, ix
- ROUTINE (#.03) Field, 16
- RPC Broker
 - Components for Delphi, 4
 - Delphi, 43
 - Login Component, 30, 31, 35, 36, 37, 39
 - Patches
 - XWB*1.1*45, 37
 - Website, xix
- RPCs, 11
 - Bypassing Security, 19
 - Create Your Own RPCs
 - Preliminary Considerations, 11
 - Process, 12

- Error Trapping, 41
- Executing, 17
- First Input Parameter (Required), 12
- Input Parameter Types (Optional), 15
- M Entry Point Examples, 15
- Online Code Samples, 20
- Registering, 19
- Relationship between an M Entry Point and an RPC, 11
- Return Value Types, 13
- RPC Entry in the REMOTE PROCEDURE File, 16
- Security, 19
 - What is a Remote Procedure Call?, 11
 - Writing M Entry Points for RPCs, 12
 - XWB GET VARIABLE VALUE, 25
- rpctest.exe, 42

S

- Security
 - Bypassing Security for Development, 19
 - How to Register an RPC, 19
 - Pass Phrase, 30, 31, 32, 37, 39
 - Phrase, 38, 39
- SECURITY KEY (#19.1) File, 19
- Security Keys
 - XUPROGMODE, 19
- SecurityPhrase Property, 30, 31, 39
- Server Property, 5, 22
- Silent Calls, 17
- Single Server Authentication, 32
- Single Signon/User Context (SSO/UC), 9
- Software Disclaimer, xiv
- Splash Screen, 23
- SplashClose Method, 23, 24
- SplashOpen Method, 23, 24
- SplVista.PAS Unit, 23, 24
- SSHPort Property, 5
- SSHPw Property, 5
- SSHUser Property, 5, 22
- SSHUseSecureConnection Property, 5
- SSO/UC, 9
- Starter Edition, 43
- strCall Method, 6, 18
- Support
 - Anonymous Directories, xx

Symbols

Found in the Documentation, xv

Syntax

GetServerInfo Function, 23

System Status Menu, 41

T

Table of Contents, x

TAG (#.02) Field, 16

TCCOWRPCBroker Component, 9

Temporary Globals

^XTMP, 40

Testing Your RPC Broker Connection, 42

TimeOut Parameter, 24

Token, 29, 31, 32, 36, 37, 39, 40

Sample, 30

Translate Function, 25

Trapping RPC Errors, 41

Troubleshooting, 41

Connections, 41

Error Trapping, 41

How to Debug Your Client Application,
41

Identifying

Handler Process on the Server, 42

Listener Process on the Server, 41

Testing Your RPC Broker Connection, 42

TRPCBroker Component, 4

Call Method, 6, 18

Connecting to an M Server, 7

CreateContext Method, 6, 19, 25

Key Properties, 5

IstCall Method, 6

Methods, 6

Properties and Methods, 4

strCall Method, 6

TXWBRichEdit Component, 9

TXWBSSOiToken Component, 10

U

Units

SplVista.PAS, 23, 24

URLs

Acronyms Intranet Website, 49

Adobe Website, xix

Glossary Intranet Website, 49

RPC Broker Website, xix

VA Software Document Library (VDL)

RPC Broker Home Page Web Address,
37

VA Software Document Library (VDL)

Website, xix

URLSTRING (#.04) Field

CALLBACKTYPE (#1) Multiple Field,
39

V

VA Software Document Library (VDL)

RPC Broker Home Page Web Address, 37
Website, xix

Value Property, 17

Version

About this Version of the BDK, 1

VistA M Server, 35, 36

VistA Splash Screen, 23

Visual Basic Language, 45

W

Web Pages

VA Software Document Library (VDL)

RPC Broker Home Page Web Address,
37

Websites

Acronyms Intranet Website, 49

Adobe Website, xix

Glossary Intranet Website, 49

RPC Broker, xix

VA Software Document Library (VDL)
Website, xix

What is a Remote Procedure Call?, 11

What Makes a Good Remote Procedure
Call?, 17

Windows Registry, 8, 22

WORD WRAP ON (#.08) Field, 13, 16, 27

Writing M Entry Points for RPCs, 12

X

XU*8.0*404, 36

XUPROGMODE Security Key, 19

XUSTATUS Menu, 41
XWB GET VARIABLE VALUE RPC, 25
XWB_DXE#.bpl File, 44
XWB_RXE#.bpl File, 44

XWBLIB
\$\$BROKER^XWBLIB, 26
\$\$RTRNFMT^XWBLIB, 27